UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Determinação de Pontos Ótimos de Inserção de Nids numa Rede Corporativa de Grande Dimensão

Gonçalo Manuel Martins Miranda

Mestrado em Engenharia Informática
Especialização em Arquitectura, Sistemas e Redes de Computadores

Trabalho de Projeto orientado por:
Prof. Doutor António Casimiro Ferreira da Costa
Eng. José António dos Santos Alegria

2019

Agradecimentos

Ao longo deste percurso tive a ajuda de muitas pessoas que sempre se preocuparam com o meu sucesso, esta secção serve como forma de agradecimento a todas elas.

Primeiramente gostava de agradecer ao Gonçalo Silva e ao Jorge Silva por tudo o que fizeram por mim, sem eles não tinha sido possível concluir este projeto. Ao longo do percurso tive vários percalços e foram eles que me ajudaram a ultrapassa-los. Sem eles inverterem o sentido da marcha o projeto poderia ter ficado estagnado num dos vários becos sem saída que foram aparecendo. Agradeço-lhes a sua disponibilidade para me ajudar, a sua boa disposição e todo o conhecimento que me transmitiram.

Agradeço ao meu orientador, Professor António Casimiro, toda a sua disponibilidade, preocupação, paciência e conhecimento transmitido ao longo deste processo. As reuniões contínuas que fomos tendo ao longo do curso deste trabalho ajudaram, em muito, a manter-me no rumo certo e a conseguir desenvolver um trabalho com qualidade.

Outra pessoa muito importante neste percurso foi o Engº José Alegria, o meu co-orientador. Agradeço-lhe toda a motivação que me deu para conseguir fazer um trabalho com a melhor qualidade possível. As suas palavras ajudaram-me em muito a superar algumas dificuldades ao longo deste trabalho e a sua crença no meu potencial também foi essencial. O seu rigor e preocupação em relação ao meu trabalho fizeram com que arranjasse motivação, em momentos por vezes difíceis, para que conseguir ter um trabalho do qual eu me orgulho!

Aos meus colegas da MEO agradeço-lhes todos os momentos que me proporcionaram ao longo deste trajeto. Obrigado à Sara Nascimento, ao Pedro Santos e ao Diego França pela paciência para me aturarem... Gostava também de agradecer a todos os meus colegas da DCY por proporcionarem todos os dias um ambiente de boa disposição, difícil de encontrar em muitos outros lugares.

Gostava também de agradecer a todos os meus colegas e amigos dos Capitals Basebol Clube. Sem eles não era possível jogar o desporto que eu amo, ainda por cima com um grupo espetacular de pessoas. O basebol foi essencial para a conclusão deste trabalho por proporcionar momentos de descontração e competição, que me permitiram desanuviar. Uma lição de vida muito importante que o basebol me ensinou foi *When life gives you bad hops make great plays!*, e por isso e por todos os outros ensinamentos que me passou vou-lhe estar eternamente grato.

Por último, gostava de agradecer à minha família, aos meus pais, ao meu irmão, aos meus avós maternos, aos meus avós paternos, ao meu tio e ao meu primo. Todos eles foram fundamentais para concluir este trabalho e sem eles nada disto era possível. Muito obrigado!

A todos aqueles que sempre acreditaram em mim.

Resumo

Uma solução eficaz de monitorização numa rede de grande complexidade é a peça fulcral para se ter um conhecimento profundo do tráfego que circula na rede. De modo a ser possível ter uma monitorização eficaz são necessários inúmeros sensores, dispersos pela rede, que permitem uma visão sobre o tráfego. Os sensores são responsáveis por capturar o tráfego da rede para que os Network Intrusion Detection Systems (NIDSs) possam depois gerar os eventos de segurança pertinentes. Os pontos onde os sensores são colocados refletem a eficácia de uma solução de monitorização, como tal o processo de determinação deve ser feito de um modo diligente, metódico, ponderado e preciso. A determinação destes pontos deve reduzir ao mínimo o número de sensores, cobrindo o máximo de tráfego possível, tendo visibilidade sobre todos os segmentos de rede, sem exceção. A execução manual desta tarefa é demorada, complexa e necessita de recursos idóneos.

Apresentamos o NIDSPlacer, um sistema capaz de determinar os pontos ótimos para a colocação de sensores de modo a cobrir o tráfego que circula de várias redes para vários destinos. O sistema foi desenvolvido com o intuito de automatizar todo o processo, reduzindo drasticamente o tempo de execução do processo em comparação com a solução manual. O resultado do processo automático é um conjunto de pontos escolhidos de forma a obter uma cobertura ótima da rede em termos da observação de tráfego. O objetivo do projeto é determinar os pontos de colocação dos sensores, de forma a monitorizar o tráfego proveniente das redes de utilizadores da MEO, com destino aos serviços críticos internos.

O sistema desenvolvido é composto por duas fases. Uma fase de descoberta inicial dos componentes da rede por onde passa o tráfego proveniente das redes de utilizadores da MEO, com destino aos serviços críticos, escolhidos pelo utilizador do sistema. A fase seguinte é uma fase de análise composta por um algoritmo, baseado no algoritmo *greedy* habitualmente associado ao problema do *minimum set cover*. O resultado desta análise é um conjunto de pontos da rede que garante a cobertura total do tráfego que circula entre as redes escolhidas, reduzindo o número de sensores ao mínimo.

Palavras-chave: Redes, Segurança, Monitorização, Descoberta da Rede, *Minimum Set Cover*

Abstract

An effective monitoring solution in a complex network is the key part to have thorough knowledge of the traffic that circulates through the network. In order to have effective monitoring inumerous sensors, scattered through the network, are needed. These sensors allow for an insight into the traffic. They are also responsible for capturing the network traffic so Network Intrusion Detection Systems (NIDSs) may generate relevant security events. The points where the sensors are placed reflect the effetiveness of a monitoring solution, as such the placement process should be done in a diligent, methodical, weighted and precise way. The process of determining these points should reduce the nummber of sensors to the minimum, covering the maximum amount of traffic as possible, having visibility of all network segments, without exception. The manual execution of this task is slow, complex and needs qualified resources.

We present NIDSPlacer, a system that is capable to determine the optimal sensor placement points in order to capture the traffic that circulates from several networks to various destinations. The system was developed with the intent to automate the entire process, drastically reducing the execution time in comparison with the manual solution. The result of the automated process is a set of points chosen in order to obtain an optimal network coverage in terms of traffic observation. The goal of this project is to determine the sensor placement points, in order to observe traffic coming from MEO's user networks, having the critical internal servers as destination.

The developed system is composed of two phases. An initial discovery phase of network components through which the traffic coming from user networks passes with destination to the critical services, chosen by the user. The next phase is an analysis done by an algorithm, based on the greedy algorithm usually associated with the minimum set cover problem. The result of this analysis is a set of network points that guarantees total coverage of the traffic that flows between the chosen networks, reducing the number of sensors to a minimum.

Keywords: Networks, Security, Monitoring, Network Discovery, Minimum Set Cover

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Garantir a segurança de uma rede de grande complexidade é uma tarefa complexa e crítica para o funcionamento correto da mesma. Um dos problemas que deve ser resolvido para garantir a segurança, é a monitorização eficaz da rede. A eficácia de uma solução de monitorização está diretamente associada à cobertura do tráfego que circula na rede, *i.e* se todo o tráfego da rede for analisado pode-se afirmar que a solução de monitorização é totalmente eficaz. A cobertura da rede é feita com sensores que permitem a captura de tráfego de uma ligação física. Os sensores são as fontes de eventos para os controlos de cibersegurança, servindo de alicerces para uma solução de monitorização focada na cibersegurança. Dessa forma, é imprescindível determinar o conjunto de pontos da rede em que o tráfego deve ser monitorizado da maneira mais eficaz. Para completar uma solução de monitorização focada na segurança são ainda necessários componentes que analisem o tráfego proveniente dos sensores, os Network Intrusion Detection Systems (NIDSs), gerando eventos de segurança que alimentam o Security Information and Event Management (SIEM) central, responsável por correlacionar os eventos recebidos.

As técnicas atuais de determinação de pontos, para a colocação de sensores, consistem em processos manuais demorosos, complexos e que necessitam de recursos idóneos, muitas vezes, difíceis de obter. Este tipo de técnicas não é escalável. Não sendo, portanto, uma escolha viável para redes de grande complexidade. Os seguintes fatores acrescem com a complexidade da rede:

- Duração do processo;
- Probabilidade de erros no resultado final, resultando numa solução de monitorização menos eficaz e eficiente;
- Necessidade de recursos qualificados;

Este conjunto de fatores demonstra que não há nenhuma vantagem na realização manual do processo de determinação de pontos para a colocação de sensores. Para além disso, o processo de determinação em si pode diferir de uma organização para a outra,

ou mesmo dentro da mesma organização, não havendo um *standard* para determinar de forma ótima, ou aproximadamente ótima, os pontos de inserção de sensores. Apenas existem alguns princípios a aplicar no processo de determinação.

O objetivo deste projeto é criar um sistema automatizado que otimize todo o processo de determinação dos pontos com vista a cobrir o tráfego oriundo de diferentes redes de utilizadores da MEO com destino a vários serviços internos críticos. Dessa forma todas as limitações do processo manual serão corrigidas, tornando o processo automatizado irrefutavelmente mais eficiente que o manual. Procuramos alcançar resultados compostos por soluções de monitorização com eficácia máxima cobrindo todo o tráfego pretendido com o menor número de sensores, tornando assim o sistema desenvolvido melhor do que qualquer processo manual com o mesmo propósito.

Este projeto foi realizado na MEO, sendo a rede interna a base deste projeto. Devido à complexidade da rede da MEO faz todo o sentido assumir que vão ser precisas algumas dezenas de sensores. Como tal é necessário ter equipamentos de rede que suportam o conjunto de sensores na prática, neste caso os equipamentos considerados são Packet Flow Switches. Estes agregam o tráfego capturado de vários sensores e depois enviam o tráfego agregado para os NIDSs.

1.1 Motivação

A determinação de pontos para a colocação de sensores é atualmente um processo manual demoroso, complexo e apenas pode ser feito por pessoas que tenham um conhecimento extenso acerca dos padrões de tráfego da rede, bem como dos seus componentes. Devido à sua complexidade, que aumenta exponencialmente com a complexidade da rede, este processo é uma tarefa bastante complicada de realizar. Dessa forma os resultados do processo, *i.e* os pontos de colocação dos sensores, podem não resultar numa solução que garanta uma eficácia e eficiência ótima, ou aproximadamente ótima, em relação á cobertura de tráfego com o menor número de sensores possíveis.

O tráfego a ter em consideração no âmbito deste projeto é aquele proveniente das redes de utilizadores da MEO, com destino aos serviços críticos a que os utilizadores acedem. Este contexto advém do facto de não ser dada a devida importância à monitorização do tráfego que circula entre os membros integrantes de uma rede privada. É possível verificar que na literatura [7] o tráfego vindo da Internet, *i.e* das redes públicas, é aquele que tem mais importância no processo de determinação de pontos, porque o nível de ameaça externo é deveras maior em relação ao nível de ameaça da rede interna. Isto não quer dizer que não existam ameaças com origem na rede interna, pelo contrário, um exemplo disso são os *worms* que entram por um computador de um utilizador interno através de um email infetado e que depois se propagam lateralmente pela rede interna afetando severamente o funcionamento de toda a rede interna, se não forem detetados e posteri-

ormente mitigados. Dessa maneira a monitorização do tráfego que circula, apenas e só, dentro da rede interna é algo deixado de parte, fazendo com que o processo manual, ineficiente e que possivelmente resulta numa monitorização ineficaz, seja considerado como apropriado. Posto isto a determinação de pontos para a inserção de sensores dentro da rede interna com o intuito de observar o tráfego que ali circula exclusivamente, é de igual importância à determinação de pontos para a observação de tráfego oriundo de redes externas. Assim, para além de proteger os ativos críticos de ameaças externas, também é necessário protegê-los de ameaças provenientes da rede interna.

Deste modo, a grande motivação deste trabalho é a criação de um sistema que possibilite a automatização de todo o processo de determinação de pontos de inserção de sensore. De forma a ter uma cobertura ótima do tráfego que circula das redes de utilizadores da MEO para os serviços internos críticos, em termos da observação de Indicators of Compromise (IOC).

1.2 Objetivos

O objetivo deste projeto é o desenvolvimento de um sistema capaz de determinar pontos, num número finito, para a inserção de sensores, de modo a cobrir a totalidade do tráfego proveniente das redes de utilizadores da MEO com destino aos serviços internos críticos. A cobertura total do tráfego permitirá observar todo o tráfego que ali circula através da sua captura utilizando os sensores. O grau de otimização da solução de cobertura do tráfego permite indicar o grau de eficiência da solução de monitorização, *i.e* uma solução mais otimizada, com menos sensores, e que cobre a totalidade do tráfego, consegue ser sempre mais eficiente, na prática, que uma solução com um maior número de sensores, menos otimizada, para a mesma rede, se cobrir exatamente o mesmo tráfego, isto porque vai haver tráfego duplicado em sensores diferentes o que vai reduzir a eficiência da solução como um todo. A eficácia da solução é indicada pela quantidade de tráfego coberto, *i.e* se todo o tráfego for coberto podemos então afirmar a total eficácia da solução de monitorização, desde que seja corretamente implementada na prática.

O sistema a desenvolver deve também ter em conta as capacidades computacionais dos Packet Flow Switches, responsáveis por enviar o tráfego que agregam de diferentes sensores para os NIDSs. Tanto os Packet Flow Switches, como os NIDSs, não são considerados no processo de determinação de pontos pelo sistema a desenvolver, sendo assim componentes externos a este processo por estarem dependentes da localização dos sensores e do tráfego coberto e não o contrário.

Em suma, para tudo isto acontecer é necessário desenhar, desenvolver, implementar e testar um sistema que vai de encontro ao objetivo especificado.

Com a construção deste sistema será possível indicar os pontos na rede onde devem ser colocados os sensores para potencializar a monitorização da rede interna ao máximo,

com uma consciência global de toda a rede e dos equipamentos que a compõem, sem ser necessária uma análise manual ineficiente, demorada e que possivelmente resulta numa solução ineficaz para toda a rede interna. Com esta identificação de pontos onde devem ser inseridos os sensores é expectável que a monitorização de eventos de segurança seja melhorada, e que os custos associados à aquisição de sensores sejam reduzidos drasticamente.

O sistema a construir deve seguir os seguintes objetivos:

- Ser escalável e compatível a todas as redes que apresentem uma estrutura semelhante;
- Ter em conta o custo associado a cada sensor;
- Ter em consideração a capacidade de processamento associada a cada sensor e outros equipamentos necessários;
- Determinação de pontos resultando numa solução de cobertura de tráfego eficaz e eficiente;
- Ter tempos de execução inferiores a uma semana;
- Melhoria significativa do desempenho e resultado em relação à solução manual

1.3 Contribuições

A principal contribuição deste projeto é a melhoria significativa do desempenho do processo de determinação de pontos de inserção de sensores na rede interna da MEO.

Para tal acontecer foi necessário criar um sistema que divide o processo em duas fases:

- A primeira fase do processo é responsável por fazer o mapeamento das redes em que circula o tráfego com origem nas redes de utilizadores da MEO, e que tem como destino os serviços críticos internos. Para criar um mapa de rede fidedigno foi criado um processo, baseado no *traceroute*, que permite a descoberta de todos os equipamentos de rede, e as respetivas ligações, entre uma máquina da rede de utilizadores, que deve correr o sistema desenvolvido, e um conjunto de serviços críticos definidos.
- A segunda fase do processo tem como encargo a determinação de pontos, na rede mapeada na fase anterior. A determinação dos pontos é feita com base no cálculo de um *rating* para todos os pontos válidos da rede, *i.e* interfaces de rede válidas. O cálculo é feito com base na cobertura de tráfego relevante pela interface. As interfaces que tenham um melhor *rating* vão sendo escolhidas à medida que aumentam as iterações do algoritmo responsável pela determinação. O algoritmo termina a sua

execução quando é atingida a cobertura máxima do tráfego relevante. A solução determina pelo algoritmo é uma aproximação da solução ótima (cobertura máxima com o menor número de interfaces), mas que garante cobertura máxima do tráfego.

O resultado final de cada execução do sistema é composto por duas soluções. Uma que garante a cobertura máxima do tráfego da rede mapeada e outra que garante a melhor relação cobertura-custo, *i.e* a solução que cobre mais tráfego com o menor número de sensores. Cada solução tem ainda uma representação visual da rede e dos pontos onde devem ser inseridos os sensores, e ainda um pequeno relatório que indica as interfaces onde devem ser colocados os sensores, bem como a largura de banda necessária para capturar o tráfego de todos os sensores.

1.4 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2 - **Monitorização de segurança em redes de grande dimensão** - O propósito deste capítulo é fazer uma breve introdução aos temas diretamente relacionados com o âmbito deste projeto.

  O primeiro tema é o funcionamento da camada de rede e da camada de ligação de uma rede de computadores, sendo este tema uma base para os outros temas. Relacionado com o primeiro tema está a deteção de intrusões na rede de uma organização. A identificação de pontos críticos numa rede é também referida neste capítulo com vários trabalhos realizados nesta área. Por último é explicado o funcionamento da plataforma de gestão de equipamentos de rede da MEO, o NNM, visto que esta é uma parte crucial do sistema desenvolvido.

- Capítulo 3 - **Arquitetura do NIDSPlacer** - Neste capítulo é descrita em detalhe a arquitetura do sistema e de cada componente.

- Capítulo 4 - **Implementação** - Neste capítulo é explicada em detalhe a implementação da arquitetura descrita no capítulo 3.

- Capítulo 5 - **Resultados e Avaliação** - Neste capítulo são apresentados os resultados da avaliação ao sistema desenvolvido.

- Capítulo 6 - **Conclusão** - Este capítulo contém um sumário do que foi feito neste projeto e apresenta algumas conclusões que podem ser retiradas do desenvolvimento do NIDSPlacer.

Capítulo 2

Monitorização de segurança em redes de grande dimensão

O sistema desenvolvido neste projeto tem como objetivo a identificação de pontos ótimos para a inserção de sensores em redes complexas, em que se incluem as redes corporativas, *i.e* descobrir e identificar pontos na rede onde devem ser colocados sensores para que a monitorização do tráfego possa ser mais eficaz, eficiente e que a solução encontrada tenha em conta o custo e capacidade de cada sensor. Posto isto, é importante estudar todas as abordagens ou princípios de abordagens já feitas sobre a inserção de sensores, onde se incluem os NIDS, numa rede de grande dimensão. É também importante explicar o funcionamento de uma rede de grande dimensão, tanto da perspetiva de segurança como de redes.

Neste capítulo é feita uma introdução a diferentes conceitos relacionados com o funcionamento e respetiva monitorização do tráfego de uma rede complexa, com um foco na sua segurança. Vão ser introduzidos conceitos de redes, bem como conceitos diretamente relacionados com a monitorização de tráfego de uma rede.

Por fim são explicadas as ferramentas usadas na construção do sistema desenvolvido, sendo que são parte integrante do sistema construído. As ferramentas servem para a construção da topologia, e para obtenção de informação relativa a *routers*, *switches* e *firewalls*.

2.1 Camada de rede e de interligação

Esta secção tem o propósito de ilustrar o funcionamento de uma rede, de modo a entender como é montada uma solução de monitorização.

Uma rede de computadores caracteriza-se por ser composta por computadores, servidores, *switches*, *routers* e outros equipamentos (*firewalls*, *hubs*, etc.). As *firewalls* são um componente opcional para o funcionamento normal da rede, visto que tem como funcionalidade a proteção de uma rede/sistema.

Um ***switch*** é um equipamento de rede composto por várias portas de rede (interfaces) ao qual se ligam *hosts* (servidores, computadores, telefones, impressoras, etc.) ou outros equipamentos de rede. A sua responsabilidade é fazer o encaminhamento de tramas *Ethernet* (ou de outro tipo, mas as mais comuns são *Ethernet*) com base no endereço Media Access Control (MAC) destino do cabeçalho da trama. Este tipo de operação está associada à L2 (camada de ligação de dados do modelo OSI [11]).

Todos os conceitos relativos a L2 e L3 são explicados com base em informação presente em [21] e [22], ambos contêm informação mais detalhada em relação ao tema.

Um endereço MAC está associado a cada interface de qualquer equipamento. Este endereço é usado numa Local Area Network (LAN), *i.e* os endereços MAC são usados para comunicações entre equipamentos do mesmo segmento de rede. Uma trama *Ethernet* é composta por seis campos: endereço MAC destino, endereço MAC origem, *tag* 802.1q (opcional, usado para identificar uma VLAN), *Ethertype* que representa o protocolo a ser utilizado (ARP, IPv4, etc.), *payload* - pacote encapsulado do tipo descrito pelo campo anterior, e no último campo um código para deteção de erros da trama.

O processo de encaminhamento de tramas feito por um *switch* é relativamente simples, na medida em que as decisões são feitas com base na tabela Content Addressable Memory (CAM) (implementação de uma tabela de encaminhamento/MACs) que guarda a informação do encaminhamento feito em cada *switch*. A construção é feita através de pedidos ARP da seguinte forma:

1. O *host* A envia um pedido ARP *broadcast* para o *host* B sabendo o seu endereço IP.

2. Assim que o *switch* ao qual está ligado o *host* A recebe este pedido, faz o encaminhamento do pedido para todas as suas interfaces (exceto aquela de onde veio o pedido).

3. Assim que algum *host* (pode ou não ser o *host* B) encontrar uma entrada da tabela CAM que corresponda ao endereço IP de B, a entrada encontrada no *host* é enviada como resposta para o *host* A.

4. A resposta segue o percurso inverso do pedido, e à medida que vai passando nos *switches*, cada um vai adicionar uma entrada à sua tabela CAM correspondente ao *host* B com o seu MAC e com a identificação da porta pela qual devem ser encaminhadas as tramas.

Estes pedidos ARP apenas ocorrem quando um *host* não tem um entrada na sua tabela ARP que mapeie o endereço IP de um *host* para o seu endereço MAC. Com este mapeamento feito, é possível enviar uma trama para um endereço IP do mesmo segmento apenas utilizando as entradas das tabelas ARP dos *hosts*, para fazer encaminhamento das tramas. No caso do endereço IP para o qual queremos enviar uma trama estar fora do segmento

de rede, os pedidos devem ser enviados para o endereço MAC do *default-gateway* da sub-rede. O *default-gateway* é um equipamento responsável por fazer o *routing* de pacotes (L3, camada seguinte no modelo OSI [11], a camada de rede) para outros segmentos de rede.

Uma LAN pode ainda ter diversas Virtual LANs (VLANs), dessa forma é possível ter mais do que um segmento de rede a partilhar a mesma infraestrutura física. Uma VLAN têm como propósito agrupar equipamentos que são do mesmo domínio de *broadcast*, na mesma VLAN, mesmo não estando conectados ao mesmo *switch*. As VLANs são utilizadas para restringir os domínios dentro de uma rede, por razões de segurança como:

- Só os equipamentos na mesma VLAN é que recebem os *broadcasts* e não todos os equipamentos de um segmento da rede;

- Os *hosts* que têm dados mais sensíveis podem estar em VLANs separadas;

Como por razões de desempenho e simplicidade de configuração:

- Apesar do maior número de domínios cada domínio vai passar a ser menor e assim os *broadcasts* podem ser controlados;

- O desenho da arquitetura de rede pode ser mais flexível visto que não há necessidade de agrupar os utilizadores por localização física e podem sim ser agrupados *e.g* por departamento;

- A reconfiguração é facilmente feita apenas alterando a que VLAN corresponde uma porta de um *switch*;

As VLANs são configuradas associando uma porta de um *switch*, que está ligada a um *host*, a uma VLAN, correspondendo assim essa porta a uma porta de acesso. Para além das portas de acesso existem ainda portas em que circula tráfego de várias VLANs, estas portas são conhecidas por portas *trunk*. Estas portas estão ligadas a outros *switches* em vez de estarem ligadas a *hosts*.

A Figura 2.1 serve para ilustrar uma arquitetura de rede com todos os conceitos descritos acima.

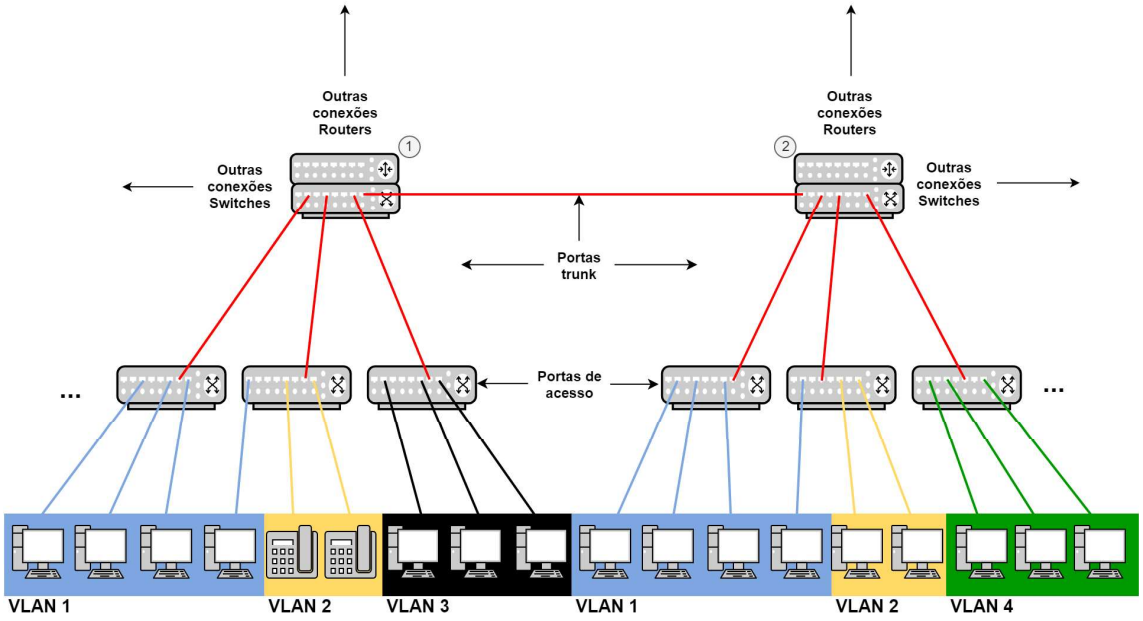


Figura 2.1: Arquitetura de uma rede

Com o uso das VLANs é possível ter diferentes segmentos de rede a partilhar a mesma infraestrutura física de L2. Os equipamentos (1) e (2) da Figura 2.1 correspondem a *switches* que também têm capacidades de *routing*. Este tipo de *switches* têm a capacidade de funcionar como um *router*, fazendo assim o *routing Inter*-VLANs. Um equipamento que tenha esta capacidade é capaz de fazer o *routing* de pacotes, encapsulados em tramas, entre duas VLANs diferentes. Desde que o *switch* tenha uma interface associada à VLAN destino ou uma interface em modo *trunk* e que depois se ligue a *switch* que tenha uma interface correspondente à VLAN e MAC destino. O *routing Inter*-VLANs pode ser feito por um *router*.

Para além das interfaces virtuais associadas às VLANs pode ainda haver outra camada virtual/lógica, a de agregação de ligações L2. Uma agregação de ligações é uma porta lógica que junta várias interfaces, tanto interfaces virtuais (correspondentes às VLANs) como aquelas associadas a portas de rede. Esta camada lógica tem o objetivo de aumentar a largura de banda para além do que uma conexão única consegue aguentar e serve também para propósitos de redundância em caso de falha de uma das ligações.

Após esta explicação dos conceitos essenciais de L2 é necessário entender a ligação entre L2 e L3, bem como o funcionamento da L3.

Posto isto, é importante explicar o que é um *router*, visto este ser o componente essencial da L3. Um ***router*** é um equipamento de rede composto por várias portas, à semelhança de um *switch*, só que a sua função é fazer o *routing* de pacotes entre diversos segmentos de rede, sendo portanto normal a comunicação com outros *routers*. Os pacotes podem ser de diversos protocolos tais como: IP, IPSec, ICMP, OSPF, RIP, etc. [22].

O protocolo mais utilizado é o IP. Um pacote IP tem dois segmentos: um cabeçalho

e outro correspondente ao *payload*. O cabeçalho tem os seguintes campos: IP origem, IP destino, Time to Live (TTL), protocolo correspondente ao *payload*, etc. Um pacote IPSec tem todos os campos de um pacote IP e mais alguns para garantir integridade, autenticidade e confidencialidade.

O *routing* de um pacote IP é feito com base no endereço IP de destino presente no cabeçalho do pacote. Após percorrer a infraestrutura associada ao seu segmento de rede chega ao *default-gateway* (*router* responsável por enviar pacotes dos *hosts* para outros segmentos de rede). Neste equipamento é verificado que o endereço IP de destino do pacote é de outro segmento, como tal é preciso fazer o encaminhamento do pacote para o próximo *router*, até chegar ao endereço IP destino.

O encaminhamento é feito em cada *router* utilizando a tabela de *routing* que cada um tem em memória. Cada entrada da tabela é composta por: rede destino e máscara de rede, *next-hop router* e interface correspondente. Quando um pacote chega ao *router* vai ser feita uma comparação entre cada rede de destino e respetiva máscara de rede com o endereço IP destino do pacote, assim que o endereço IP destino pertencer à rede de uma entrada da tabela, é selecionada a interface de saída e é feito o encaminhamento do pacote para o *router* indicado através da interface indicada, isto depois de decrementar, em uma unidade, o TTL associado ao pacote. Se não for encontrada nenhuma entrada correspondente, o pacote é descartado.

A tabela de *routing* pode ser construída de diversas maneiras:

- **Rotas de redes diretamente conectadas** - Este tipo de rotas representam redes que estão associadas a interfaces do *router* e que portanto estão diretamente conectadas ao *router* e não precisam de passar por outro equipamento de L3

- **Rotas estáticas** - Estas rotas são definidas manualmente no *router* e são usadas quando o *router* não consegue fazer o *routing* dos pacotes para a rede definida, *i.e* não existe uma conexão direta para aquela rede.

- **Rotas dinâmicas** - Estas rotas são aprendidas por um *router* através de protocolos de *routing* (IGRP, RIP, BGP, EIGRP e OSPF). Um protocolo de *routing* é usado para a troca de informação de *routing* entre *routers*.

  Este tipo de protocolos são dinâmicos porque aprendem as rotas através de outros *routers*, sendo que a tabela de *routing* pode ser alterada a qualquer momento. Com o uso destes protocolos é possível escolher rotas diferentes em caso de falha de uma ligação, não precisando assim de uma configuração manual que tenha isso em conta. A única configuração que é precisa fazer são as redes a anunciar aos outros *routers* (mesmo até esta configuração pode ser dinâmica com o uso de protocolos de *routing* dentro de uma rede com inúmeros *routers*, os protocolos intra-AS), estas redes têm de estar diretamente conectadas ao *router*. O funcionamento destes protocolos de

routing está para além do âmbito deste trabalho, não sendo pertinente explicar estes protocolos em detalhe.

2.2 Deteção de Intrusões

Sendo que este projeto está inserido na área de segurança informática, é importante contextualizar o seu âmbito, apesar do foco ser a monitorização de uma rede.

Nesta secção são introduzidos os conceitos mais pertinentes na área para que seja possível entender melhor o enquadramento deste projeto na área de segurança informática.

Posto isto, é de extrema importância entender os principais conceitos relacionados com cibersegurança. Um dos principais conceitos é o de **vulnerabilidade** que se define como: característica ou fraqueza específica que torna uma organização ou ativo aberta a uma exploração por uma dada ameaça ou suscetível a um dado risco. O ato de **ataque** é uma tentativa de ganhar acesso não autorizado a serviços, recursos, informação ou até comprometer a integridade do sistema, sendo para isso necessário a exploração de uma vulnerabilidade presente no sistema.

Os ataques podem ser feitos de duas formas distintas, ativa ou passivamente. Os **ataques ativos** são ataques perpetrados por uma fonte de ameaça intencional que tenta alterar o sistema, recursos, dados ou a operacionalidade de um sistema. Os **ataques passivos** são semelhantes aos ativos na medida em que existe intenção de entrar num sistema, apenas se diferenciam pelo facto de tentar retirar informação sobre o sistema e não tentam alterar o estado do sistema.

Após um ataque bem sucedido o sistema fica num estado erróneo o que se pode classificar como uma **intrusão**. Nesta fase existem,normalmente, mecanismos para prevenção ou deteção de intrusões para que o ataque realizado possa ser detetado e/ou mitigado. Aqui podemos encontrar diferentes tipos de sistemas que têm como objetivo a prevenção de intrusões. Existem dois tipos de sistemas que são utilizados por toda a indústria, estes dividem-se depois em subcategorias. Esses sistemas são os **Intrusion Detection Systems (IDSs)** e os **Intrusion Prevention Systems (IPSs)**.

Um IDS é um sistema que monitoriza o tráfego da rede e que gera alertas quando encontra comportamento que considera malicioso. Os IDSs dividem-se em duas subcategorias os **Host IDS (HIDS)** e os **Network IDS (NIDS)**. Os HIDSs correm em hosts individuais (servidores, computadores, etc.) ou em equipamentos que estão na rede. Já os NIDSs geram eventos de pontos estratégicos da rede, de modo a monitorizar o tráfego mais crítico. Os IPSs cumprem funções semelhantes aos IDSs identificando assim atividade maliciosa, guardando logs sobre essas atividades, informando sobre tais atividades e ainda tentam bloquear ou parar a atividade maliciosa identificada. Esta última funcionalidade é a que diferencia um IPS de um IDS.

Sendo que o âmbito deste trabalho é a colocação de sensores que capturam tráfego

de uma rede para depois os NIDSs gerarem eventos de segurança, é pertinente explicar o seu funcionamento e o seu propósito mais detalhadamente. Posto isto, um NIDS pode ser conectado de diversas maneiras para que consiga capturar e monitorizar o tráfego. A primeira possibilidade é conectar um NIDS diretamente a um *switch ou router*, tendo este equipamento de ter uma ou várias portas que façam *mirroring* (duplicação do tráfego). No caso dos *switches* podem existir portas específicas para esse efeito, as Switched Port Analyzer (SPAN). Estas portas servem para recolher o tráfego que passa numa interface de rede, portanto é só necessária uma ligação entre uma SPAN do *switch* e um NIDS. Com estas portas é possível obter uma cópia de todos os pacotes que sejam observados numa porta ou numa determinada Virtual Local Area Network (VLAN), de modo a serem analisados. Um dos benefícios desta opção é o facto de em caso de falha do NIDS não há perturbação do tráfego da conexão original, dado que ele esta diretamente ligado ao equipamento por uma ou mais SPANs que não interferem com o tráfego original.

Para além desta opção, através do uso de *hardware* específico, é possível ter um NIDS como um *software* que corre dentro do sistema operativo do *switch ou router* e que vê todo o tráfego que ali passa, mas não é uma solução muito utilizada por não ser muito prática.

A segunda opção é o uso de vários **Test Access Point (TAP)** ou ***Packet Flow Switches*** que agregam o tráfego de diversos equipamentos de rede (*routers*, *switches*, etc.), através de ligações a partir de portas *mirror* ou através do uso de sensores,*e.g splitters*, que capturam o tráfego de uma ligação. Tanto os TAPs como os *Packet Flow switches* não interferem de qualquer maneira com o tráfego que circula nas ligações que passam por eles. A sua responsabilidade é fazer uma cópia dos pacotes que circulam pelas ligações, nas portas respetivas, e depois enviar essa cópia para um NIDS que está ligado às suas portas de monitorização, que apenas enviam tráfego. Esta opção é aquela considerada para a implementação de uma solução de monitorização, neste projeto.

A terceira opção é a colocação dos NIDSs *inline*, ou seja, colocar um NIDS diretamente conectado a uma ligação da rede, em que todo o tráfego de rede vai passar diretamente por ele. Esta solução traz várias desvantagens sendo uma delas o facto de a rede estar dependente do funcionamento NIDS, quer dizer, se o NIDS falhar, a rede deixa de funcionar, apesar do NIDS não ser um equipamento essencial para o funcionamento normal da rede.

Um NIDS pode detetar intrusões de duas maneiras (ou ambas), *signature-based* ou *anomaly-based*. Um NIDS *signature-based* procura padrões suspeitos nos pacotes (*signatures* - assinaturas) que representem intrusões de rede já conhecidas. Já um NIDS *anomaly-based* usa como base o comportamento do sistema no seu estado normal, para detetar atividade suspeita no tráfego. Esta técnica associa-se normalmente a aprendizagem automatizada, pois implica uma fase de aprendizagem para o NIDS saber qual o estado normal do sistema, de modo a não indicar falsos positivos, *i.e* considerar que está

na presença de uma intrusão quando na verdade se trata de um comportamento normal do sistema. Para além de falsos positivos o sistema pode também apresentar falsos negativos, quer dizer, não detetar um intrusão no sistema e identificar essa intrusão como um comportamento normal.

Ambos os tipos de NIDSs têm vantagens e desvantagens. Os NIDSs *signature-based* são normalmente mais rápidos, conseguem menos falsos positivos e não necessitam de uma fase de aprendizagem. A sua principal desvantagem vem do facto de não detetarem ataques *zero-day*[1]porque ainda não reconhecem a sua assinatura. Os NIDSs *anomaly-based* são mais difíceis de pôr em funcionamento, devido à sua configuração e fase de aprendizagem do sistema, mas conseguem detetar novos ataques que se diferenciem do comportamento normal do sistema. Desde que tenham uma fase de aprendizagem com tráfego representativo do estado normal do sistema, vão conseguir detetar uma intrusão em condições normais.

Neste trabalho os NIDSs considerados são NIDSs *signature-based open-source* Suricata e a suas fontes de pacotes da rede vão ser *Packet Flow Switches* que recebem o tráfego a partir de *splitters* colocados em várias ligações físicas a determinar pelo sistema desenvolvido.

Todos os conceitos aqui referidos são explicados em mais detalhe num *survey* sobre os IDSs. [23]

2.3 Identificação de pontos críticos em redes

Nesta secção são apresentados trabalhos relacionados com a inserção de NIDSs em redes com ativos críticos (*e.g* servidores DNS, email, Active Directory - AD e BDs). São também apresentados conceitos relacionados com a cobertura de redes e com teoria de grafos, mais concretamente, conceitos relacionados com o princípio da centralidade.

Um dos trabalhos existentes sobre a colocação de NIDSs, dá conselhos sobre a sua colocação na rede de uma grande organização [7], mas não apresenta um conceito de uma possível solução para o problema e muito menos uma solução automatizada que funcione em redes com qualquer estrutura. Neste trabalho é possível perceber que para colocar um NIDS numa rede é necessária uma análise da topologia de rede da organização. Esta análise entende-se como a identificação de diferentes pontos como os pontos de acesso à Internet, os pontos de ligação à Wide Area Network (WAN) e os pontos de acesso remoto, *e.g* VPN. Para além desta análise é necessária a identificação dos seus serviços críticos. Com os resultados desta análise o administrador de sistemas responsável, vai decidir onde colocar os NIDSs tendo em conta a informação recolhida e os recursos disponíveis.

Outro dos trabalhos existentes relacionado com o tema da colocação de sensores, que possibilitem o funcionamento de IDSs numa rede, apresenta uma solução para a colocação

[1] *Zero day* - Ataque desconhecido pelo fabricante e pela comunidade de cibersegurança

de sensores baseada em modelos de grafos de ataque [25]. A solução apresentada é aproximadamente ótima, sendo que apenas é necessário um número mínimo de sensores para a recolha de tráfego. Utilizando grafos de ataque gerados a partir de uma análise da configuração da rede, topologia, dispositivos que limitam a conectividade como *firewalls*, serviços de vulnerabilidades, etc. Fazendo depois uma correspondência entre essa análise e ***exploits***[2]de ataque conhecidos.

A partir deste grafo de ataque em conjunto com a topologia de rede, é possível identificar o conjunto aproximadamente ótimo de pontos na rede onde devem ser inseridos sensores, de modo a ter uma cobertura máxima das rotas identificadas pelos grafos de ataque.

O problema da cobertura de uma rede com sensores equivale a um problema clássico no campo da informática, o *minimum set cover* [20]. Uma solução para este problema é a utilização de um algoritmo *greedy* (que escolhe os conjuntos de maior tamanho e com elementos menos frequentes, até conseguir uma solução) de tempo polinomial que resolve o problema do *minimum set cover* [28] com uma aproximação da solução ótima. Desta forma é possível identificar os pontos da rede onde devem ser colocados sensores para a captura de tráfego. A topologia da rede é aqui tida em conta na medida em que cada equipamento de rede tem a ele associado os caminhos do grafo de ataque que passam por ele. Assim o algoritmo escolhe os equipamentos de rede para que seja possível cobrir todos os elementos do grafo, conseguindo uma solução ótima para a inserção de sensores, tendo como base o grafo de ataque.

O trabalho descrito em [7] apresenta uma ideia de colocação de NIDSs relacionada com as zonas mais críticas de uma rede (zonas de entrada para a rede interna e zonas onde estão os serviços críticos), aconselhando à inserção de NIDSs nessas zonas. Já o segundo trabalho, descrito em [25], acrescenta outros fatores a ter em conta na colocação de NIDSs. Dando mais importância à cobertura ótima da rede com diversos fatores a ter em conta, sendo um deles a criticidade de algumas zonas da rede.

A identificação de pontos críticos em redes/grafos relaciona-se diretamente com o conceito de centralidade da teoria de grafos [9]. Diferentes métricas de centralidade são usadas para identificar os vértices mais importantes dum grafo. As diferentes métricas de centralidade habitualmente referidas na literatura [18] são as seguintes:

- Baseada no grau de cada nó
- Baseada na proximidade de um nó em relação a todos os outros nós
- PageRank [26]

Uma outra possível métrica relacionada com a importância de nós numa rede é a aplicação do modelo Barabási-Albert [8]. O modelo diz que cada nó tem a ele associado

[2]*Exploit* - Técnica usada para quebrar a segurança da rede ou sistema de informação em violação da política de segurança

uma probabilidade de um novo nó se conectar a ele, essa probabilidade é definida pela sua conectividade, à semelhança da métrica de centralidade baseada no grau de cada nó. Este modelo não é associado ao conceito de centralidade por ser utilizado em redes que seguem uma distribuição *power-law*. A topologia de rede da Internet tem uma relação com a *power-law* [14], como tal o modelo Barabási-Albert pode ser associado a uma rede com uma estrutura semelhante à da Internet. Uma rede corporativa (Intranet) tem uma estrutura semelhante à da Internet, sendo desta forma possível aplicar o modelo a qualquer rede corporativa.

Com a aplicação destas métricas a um grafo, é possível ordenar todos os vértices pela sua importância.

O conceito de centralidade não está diretamente ligado com a cobertura ótima do tráfego que circula numa rede, que é o objetivo deste trabalho. Apesar de não estar diretamente relacionado com o objetivo do trabalho, é importante mencionar estes conceitos por terem sido estudados como possibilidade para a base do algoritmo de identificação de pontos ótimos para a colocação de sensores.

2.4 Network Node Manager (NNM)

O NNM é uma plataforma de gestão de equipamentos de rede. A sua função é agregar informação sobre equipamentos de rede. As informações vão desde dados de performance de cada equipamento (estado do equipamentos, estado das interfaces, etc.), a dados da topologia da rede que são representados pelas conexões entre os equipamentos.

Esta plataforma foi utilizada neste projeto por concentrar nela todos os metadados dos equipamentos de rede da MEO, sendo que essa informação já se encontra filtrada e organizada, evitando assim pedidos diretos aos equipamentos e o posterior tratamento desses dos dados obtidos, já que essa informação se encontra disponível via API.

A obtenção dos metadados de cada equipamento é feita pelo NNM através de pedidos SNMP. Estes pedidos são feitos pontualmente, tendo como propósito obter informações relativas às interfaces e ao equipamento em si. As informações obtidas sobre cada interface do equipamento são: nome, tipo, velocidade, índice da porta, *alias* e endereço MAC. São também obtidas informações em relação ao próprio equipamento: marca, tipo, nome, endereço IP de gestão, localização, entre outras. Os dados e meta-dados dinâmicos são obtidos da mesma forma. Estes dados são depois utilizados para fazer a monitorização dos equipamentos (número de pacotes numa interface, tempo de atividade do equipamento, etc.) e para fazer as ligações da topologia de rede (tabela de *routing*, BD de encaminhamento, informação do Link Layer Discovery Protocol - LLDP, etc.).

A BD da topologia é construída, numa fase de descoberta, feita periodicamente, é dividida da seguinte maneira:

- Construção da topologia de L3: Obtenção da tabela ARP e tabela de *routing*, e com

essa informação são feitas as ligações entre os equipamentos.

- Construção da topologia de L2: Obtenção das ligações físicas com os equipamentos vizinhos através do LLDP, CDP e outros protocolos proprietários de descoberta explicados nas secções seguintes deste capítulo. Para além destes protocolos de descoberta de ligações, o NNM obtém ainda a BD de encaminhamento, que indica qual porta de rede deve ser utilizada quando é feito encaminhamento de uma trama para um certo endereço MAC destino.

A Figura 2.2 ilustra o funcionamento do sistema, num nível abstrato, bem como, a invocação de funções via API que permitem a obtenção de informação armazenada pelo NNM.

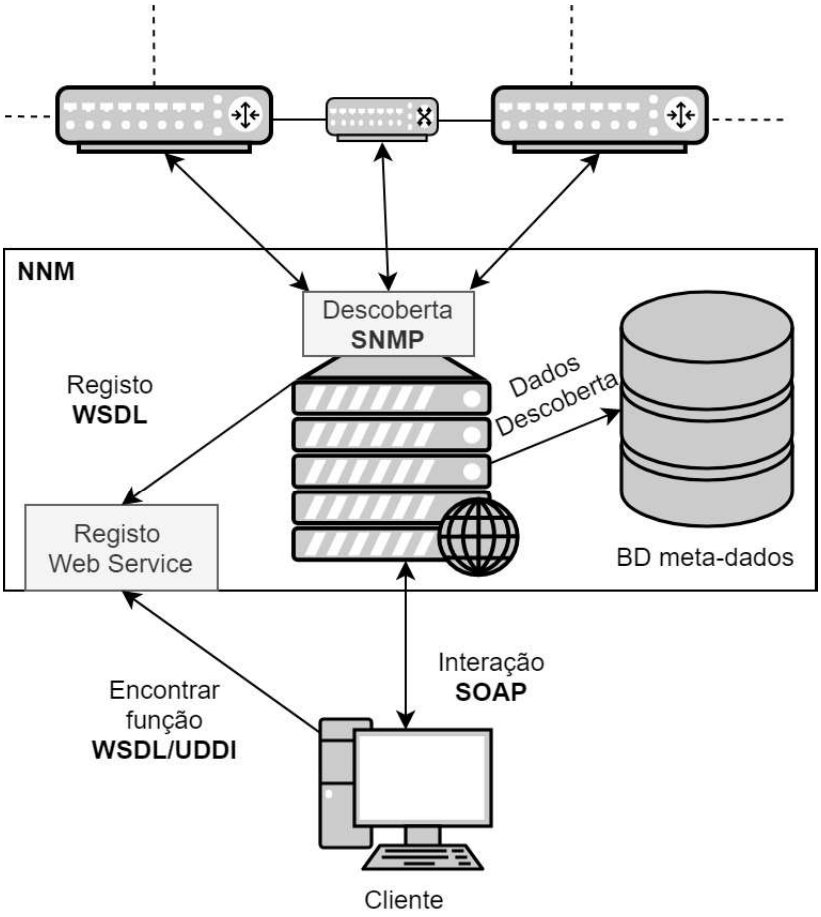


Figura 2.2: Arquitetura NNM

A API do NNM pode ser acedida via serviços web. Os serviços disponibilizados pelo NNM são indicados em vários ficheiros WSDL armazenados pelo NNM. A invocação de uma função dum serviço implica uma primeira fase de descoberta da função e só depois é possível invocar essa função via API, tal como se pode ver na Figura 2.2. As funções disponibilizadas vão buscar informação às BDs internas que são populadas pela fase de descoberta, feita periodicamente.

Nas subsecções seguintes o SNMP, LLDP e CDP são explicados em mais detalhe.

2.4.1 Simple Network Management Protocol (SNMP)

O SNMP é um dos protocolos de gestão de redes mais utilizados desde a sua criação. É particularmente útil, porque permite obter informação de estado dos equipamentos que o tenham ativado. Os equipamentos de rede correm um agente SNMP, que está à escuta de pedidos SNMP e que responde de volta com a informação pedida.

Este protocolo é bastante utilizado por ser particularmente simples de utilizar e configurar. Permite obter informação (leitura de valores), bem como, a configuração de serviços como o Dynamic Host Configuration Protocol (DHCP) (escrita / alteração de valores).

Para obter a informação de um agente SNMP, o agente tem de ter diferentes ficheiros Management Information Base (MIB) que normalmente estendem as MIBs mais básicas. Cada ficheiro deste tipo está estruturado em árvore e é composto por diferentes objetos identificados por Object Identifiers (OIDs). Cada objeto desses vai ter associado um valor que o representa. Os OIDs do topo da árvore representam as organizações de estandardização como a International Organization for Standardization (ISO).

O protocolo SNMP recolhe informação utilizando operações como o *get, getnext, getbulk* (introduzido na segunda versão do protocolo), tal como se pode ver na Figura 2.3, as duas primeiras operações permitem obter um valor de um dado OID e a última obtém vários OIDs de uma vez. Por sua vez as respostas aos pedidos *get* são de *get-response* e trazem a resposta ao pedido.

Para além da recolha de informação através de pedidos SNMP dirigidos para o agente, o agente pode também enviar informação para um servidor de gestão desde que seja configurada uma *trap* SNMP no equipamento. Uma *trap* permite que seja obtida informação sem que haja um pedido para obter tal informação. Pode ser útil quando por exemplo, um valor passa um determinado limite pré-estabelecido que se possa considerar passível de gerar um alarme. O funcionamento do protocolo está ilustrado na Figura 2.3.

Este protocolo é parte essencial do NNM pois a maioria informação retirada dos equipamentos presentes na plataforma é obtida através de pedidos SNMP.

O SNMP tem três versões. A primeira (v1) fazia autenticação de clientes apenas com base numa *community string*, transmitida em claro, e que apenas precisava de ser igual aquela que estava configurada no agente. A segunda versão (v2c) introduziu a operação *get-bulk* e alguns melhoramentos a nível de performance, também introduziu um novo mecanismo de segurança que foi considerado demasiado complexo e como tal, continuaram-se a utilizar as comunidades em claro. Esta é a versão mais utilizada atualmente, através do uso de comunidades como a *public* e *private*. Já a última versão lançada, a terceira, não faz qualquer alteração ao protocolo base da segunda versão, mas tem como maior preocupação o aumento da segurança, com a adição de técnicas de criptografia.

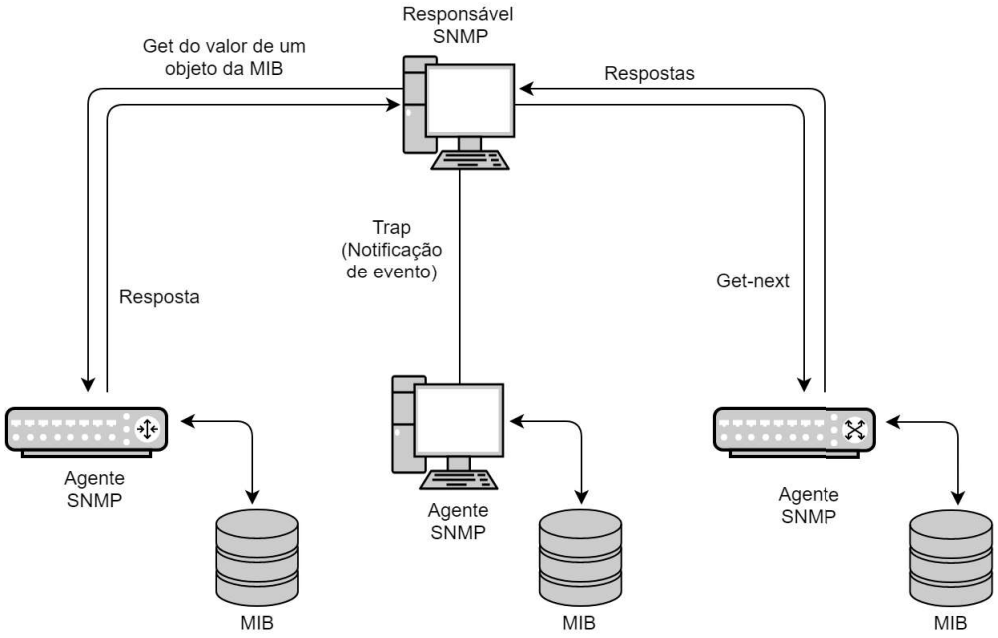


Figura 2.3: *Flow* dos pedidos SNMP

Esta versão oferece autenticação da fonte dos pedidos SNMP, encriptação dos pacotes (para garantir confidencialidade), garante a integridade das mensagens e ainda fornece controlo de acesso a configurar no agente, este controlo determina quais utilizadores têm autorização para aceder a qual MIB, objeto, etc.

2.4.2 Link Layer Discovery Protocol (LLDP)

O LLDP é o protocolo *standard* de descoberta L2 que pode ser utilizado por todos os equipamentos que partilhem a mesma infraestrutura física L2. Este protocolo é utilizado para os equipamentos de rede do mesmo segmento de rede anunciarem as suas identidades, capacidades e vizinhos na sua rede local (LAN). O LLDP armazena a informação recolhida num ficheiro MIB no dispositivo. Esta informação pode ser recolhida utilizando o SNMP, e pode conter nomes de VLANs, capacidades do sistema (switching, routing, etc.), agregações de ligações ou o endereço IP de gestão.

2.4.3 Cisco Discovery Protocol (CDP)

Ao contrário do LLDP, o CDP é um protocolo proprietário da Cisco, enquadrando-se também na L2. É utilizado para partilhar informação sobre equipamentos Cisco. Essas informações podem incluir o Sistema Operativo, endereços IPs de equipamentos, interfaces dos vizinhos que têm conexões para ele, etc. À semelhança do LLDP, o CDP também utiliza as MIBs como unidade de armazenamento de informação.

À semelhança do CDP existem outros protocolos proprietários de descoberta L2. O Extreme Discovery Protocol (EDP) é um exemplo de outro protocolo proprietário, usado

em equipamentos de rede da Extreme Networks.

2.5 Sumário

Este capítulo introduziu conceitos relacionados com monitorização de uma rede, com foco na segurança. Para entender melhor a monitorização de uma rede começámos por explicar o funcionamento da L2 e a sua ligação com a L3. A monitorização focada na segurança é descrita no capítulo seguinte, com uma maior incidência no funcionamento dos NIDSs.

Para ser possível criar uma solução de monitorização eficaz é necessário saber onde devem ser colocados os sensores de captura/inspeção de tráfego. Uma solução eficaz de monitorização está relacionada com os pontos críticos de uma rede e para entender melhor esse ponto foi necessária uma análise de trabalhos feitos neste campo, isto para que o projeto desenvolvido tenha em conta o trabalho já feito no passado.

A plataforma interna da MEO essencial neste projeto é o NNM, visto que agrega toda a informação de todos os equipamentos de rede internos, como tal foi essencial explicar o seu funcionamento.

O próximo capítulo apresenta a arquitetura do sistema desenvolvido, o NIDSPlacer.

Capítulo 3

Arquitetura do NIDSPlacer

Neste capítulo é abordada a arquitetura do sistema desenvolvido, o **NIDSPlacer**. Este sistema foi criado para resolver o problema da colocação de sensores que permitam a captura do tráfego, e que juntamente com os *packet flow switches* e NIDS formam uma solução de monitorização com foco na segurança. A proposta apresentada é um sistema que tem como função indicar os pontos onde devem ser colocados sensores, com vista a garantir uma cobertura ótima do tráfego proveniente da rede interna de utilizadores da MEO com destino aos serviços críticos presentes nos diversos datacenters internos.

Com esta garantia é possível ter uma cobertura ótima dos IOCs a serem observados na rede em que circula o tráfego pertinente.

3.1 Requisitos do Sistema

Qualquer sistema de *software* deve ter um conjunto de características que deve cumprir. Este conjunto é normalmente definido através de requisitos do sistema. No caso do NIDSPlacer os requisitos estão relacionados com dimensão e complexidade da rede a analisar e com o número de serviços críticos a analisar.

Os requisitos que o sistema deve cumprir são os seguintes:

- **Escalabilidade -** Deve ser escalável a redes de grandes dimensões.
- **Modularidade -** Os componentes do sistema devem ser módulos, havendo assim uma independência entre eles. Dessa forma é possível alterar e perceber com maior facilidade a implementação de cada um dos componentes.
- **Versatilidade** - Qualquer mudança nos meta-dados relativos a servidores e equipamentos de rede não deve implicar uma mudança no código fonte.
- **Usabilidade -** O sistema deve ser simples e prático de usar. As configurações necessárias para pôr o sistema em funcionamento devem ser simples de entender, com a documentação correta.

- **Compatibilidade** - O sistema deve ser compatível com qualquer estrutura de rede. Para além disso o sistema deve também ser compatível com qualquer equipamento de rede, seja de que marca for.
- **Desempenho -** Os resultados produzidos pelo sistema, tanto da descoberta de rede como da colocação dos sensores, devem representar corretamente qualquer estrutura de rede.
- **Eficiência -** O sistema deve apenas a realizar as tarefas necessárias para produzir o melhor resultado possível, cumprindo os restantes requisitos.

3.2 Arquitetura da Solução

A Figura 3.1 representa a arquitetura do sistema desenvolvido de uma forma abstrata. O sistema apresenta uma arquitetura de software em *pipeline*, visto que todos os módulos dependem do módulo anterior para poderem cumprir a sua função. Esta dependência deve-se ao facto de serem precisos dados provenientes do módulo anterior, sendo estes dados passados de um componente para o outro utilizando *buffers* em memória partilhados entre eles. O sistema é então composto por 3 fases/módulos. A separação do sistema em fases/módulos respeita o requisito de **modularidade**.

A primeira fase, **DiscoveryEngine**, tem apenas como responsabilidade a descoberta da rede entre as redes de utilizadores definidas e os *datacenters* que alojam servidores com os serviços indicados como *input*. Para fazer essa descoberta, tira partido de meta-informação armazenada em BDs internas que contém informação sobre equipamentos de rede e servidores.

A fase seguinte, **PlacementEngine**, é responsável por determinar os pontos ótimos para a colocação de sensores na rede descoberta pelo DiscoveryEngine na fase anterior.

Já a última fase é meramente uma camada de abstração entre a complexidade do processo de determinação dos pontos para colocação de sensores e o output final mostrado ao utilizador. Este processo é responsável por juntar toda a informação de forma percetível e apresentável para o utilizador do sistema.

3.2.1 DiscoveryEngine

O **DiscoveryEngine** corresponde à 1ª fase da *pipeline* e tem um só objetivo: obter a topologia e composição física da rede interna entre diversas regiões dessa mesma rede (*VPN*, rede por cabo, diferentes *datacenters* físicos, etc.). As redes individuais a incluir no mapeamento da rede são definidas num ficheiro de configuração. A exclusão de algumas regiões, feita pelo sistema, tem como propósito a versatilidade de configuração, um exemplo é a exclusão de redes críticas em que não podem ser colocados sensores, sem causar grande perturbação na rede ou simplesmente porque o utilizador do sistema não

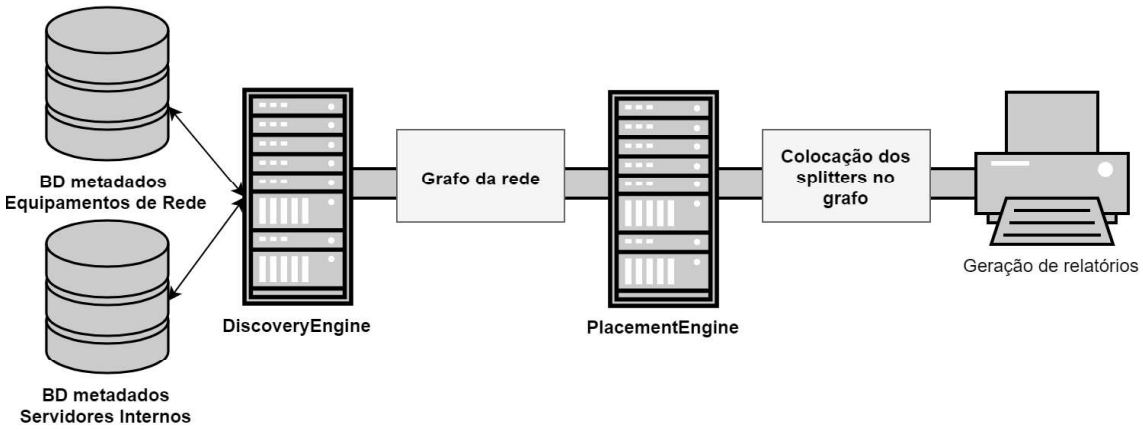


Figura 3.1: Arquitetura do NIDSPlacer

pretende colocar sensores em determinadas redes. A topologia de rede descoberta define várias redes L2 unidas. Os nós representam equipamentos de rede, por sua vez cada nó tem sub-divisões que representam as suas interfaces de rede. As ligações entre nós são feitas de uma interface para outra, representando uma ligação física L2 existente. Cada ligação é um elemento do conjunto que serve de input para o **PlacementEngine**.

Este componente é o motor do sistema, na medida em que não depende de quaisquer outros módulos do sistema e sem ele não haveriam dados para processar pelos outros módulos, deixando o sistema obsoleto. Este módulo poderia ser utilizado por si só, para fazer uma descoberta da rede, obtendo assim informações em relação à estrutura da rede.

Este módulo tem dois componentes que permitem a obtenção de metadados, sendo ambos camadas de abstração para a obtenção de informação presente em componentes externos e independentes ao sistema. Um dos componentes trata de todas as operações associadas aos metadados de equipamentos de rede, o outro trata dos metadados associados aos serviços que correm nos servidores, mas também tem algumas informações alusivas aos equipamentos de rede.

A arquitetura deste módulo surgiu como uma de quatro opções possíveis descritas de seguida.

Uso de uma ferramenta de mapeamento da rede

A primeira possibilidade surgiu como a utilização de uma ferramenta com a capacidade de mapear redes, o Network Topology Mapper da SolarWinds. Esta ferramenta teria de ser corrida sempre que o sistema fosse corrido. Assim o processo de descoberta seria síncrono ao sistema porque o mapeamento da rede seria feito a pedido do sistema, sendo que este só avança quando obtiver uma resposta válida da ferramenta.

Uma ferramenta deste tipo tem um tempo de execução bastante demorado, devido à quantidade de informação que obtém por cada equipamento (a grande maioria desta informação é desnecessária para o propósito do NIDSPlacer) e por fazer uma descoberta

não dirigida. Se a descoberta fosse dirigida, *i.e* se fosse possível mapear o caminho L2/L3 percorrido entre dois IPs, sub-redes *e.g* através do uso dos *default-gateways*, ou apenas o caminho entre a máquina que faz o *scan* e um conjunto de endereços IP dados como destino, teria sido possível utilizar esta opção.

Sem essa garantia o esforço necessário para desenvolver uma solução com base numa ferramenta destas seria excessivo face às outras alternativas.

Desenvolvimento de um processo de mapeamento da rede através de SNMP

Outra possibilidade era a criação de um processo de descoberta da rede independente deste módulo e do sistema. Este processo mapearia a rede, contactando diretamente cada um dos equipamentos de rede encontrados, à medida que se iam efetivando as ligações entre eles, usando SNMP. Este processo é bastante complexo e por si só, já é um projeto de grandes dimensões, como tal não era uma solução viável.

Uso de ferramenta que tira partido de informação do mapeamento da rede

Como a possibilidade anterior foi rapidamente descartada, tentou-se arranjar outra solução. Essa solução passava pela utilização de uma ferramenta que tirava partido da informação de um repositório de dados, neste caso era o NNM que tinha uma funcionalidade que permitia obter um caminho de rede entre dois equipamentos. Um repositório deste tipo é alimentado por ferramentas de descoberta e mapeamento da rede. Dessa forma era possível desacoplar o processo de obtenção de informação (que é feito de forma assíncrona em relação a este módulo) do processamento e da análise. Esta opção deixou de ser uma escolha válida pela falta de mapeamento correto de muitos caminhos de rede no NNM, visto que o *output* final do sistema não iria transparecer a realidade da rede.

Desenvolvimento de um processo de mapeamento da rede baseado no *traceroute*, que usa informação dos equipamentos de rede armazenada num repositório

A última possibilidade explorada foi a criação de um processo de descoberta baseado no *traceroute* [6]. A utilização do *traceroute* tem como propósito dirigir o processo de descoberta na direção do tráfego entre dois pontos (o endereço fonte está associado a uma interface da máquina onde é corrido o *traceroute* e o endereço destino é indicado como *input*). Os endereços IP que aparecem no *output* do *traceroute* são endereços associados a interfaces presentes em equipamentos de rede. Com esta possibilidade o processo de descoberta torna-se síncrono ao sistema, à semelhança de qualquer outro pedido de descoberta da rede.

Desta forma é possível identificar o equipamento e a interface associada a um endereço através da utilização de um repositório com informação dos equipamentos de rede, neste caso o repositório é o NNM, por agregar informação sobre todos os equipamentos de rede

da MEO. Com cada uma das interfaces identificadas é possível identificar as ligações físicas relativas a cada uma das interfaces. Estas ligações são identificadas através de informação LLDP, CDP ou de outros protocolos de descoberta proprietários. Desde que esta informação se encontre armazenada num repositório. Como tal é possível desenvolver um processo que agrupe toda esta informação num só mapa de rede.

A arquitetura escolhida para este módulo foi então a opção do processo de mapeamento da rede com base no *traceroute* por providenciar uma maior independência em relação a ferramentas de descoberta da rede que poderiam trazer complicações desnecessárias. A primeira opção foi ainda explorada mas trazia mais desvantagens e complicações desnecessárias em relação à opção escolhida e foi portanto excluída após um processo de experimentação. A segunda opção foi rapidamente excluída por ser impraticável devido à sua complexidade e por isso, iria envolver um esforço adicional necessário para um processo que não é o âmbito deste projeto. Já a terceira opção foi testada durante um longo período de tempo, sendo considerada a principal opção para a arquitetura deste módulo durante grande parte do projeto, mas ao longo do tempo tornou-se percetível que não iria produzir resultados minimamente aceitáveis devido à falta de mapeamento correto da rede. Isto resultava num funcionamento deficiente da ferramenta que descobria os caminhos de rede.

A Figura 3.2 ilustra o processo de descoberta de rede, feito individualmente, para cada servidor identificado e rede indicada no ficheiro de configuração. O processo começa com os dados provenientes de um *traceroute*. Os argumentos do *traceroute* que se diferenciam em todos os *scans* feitos são o servidor e a interface da qual é feito. A interface corresponde a uma das redes indicadas no ficheiro de configuração.

O *input* dado para o algoritmo do **PlacementEngine** é o conjunto de interfaces descobertas. Cada interface por sua vez tem a ela associada um conjunto de rotas cobertas.

3.2.2 PlacementEngine

Este módulo é a 2ª fase do processo que compõe a *pipeline*, sendo esta a fase de análise da rede e determinação da distribuição de sensores na rede. Esta é a peça essencial do sistema. Todo o processo de decisão dos locais de inserção dos sensores é feito nesta fase. A sua função é a determinação de uma solução que contenha os pontos, da rede mapeada pelo **DiscoveryEngine**, onde devem ser colocados os sensores que possibilitam a captura de tráfego. Os sensores tidos em consideração foram *splitters* de fibra ótica, mas qualquer tipo de sensor que permita a captura de tráfego que funcione de maneira semelhante pode ser utilizado para implementar a solução encontrada.

Os *splitters* são equipamentos que separam uma ligação de rede em duas ligações independentes. Este tipo de *splitters* separa um raio de luz, proveniente de um cabo de

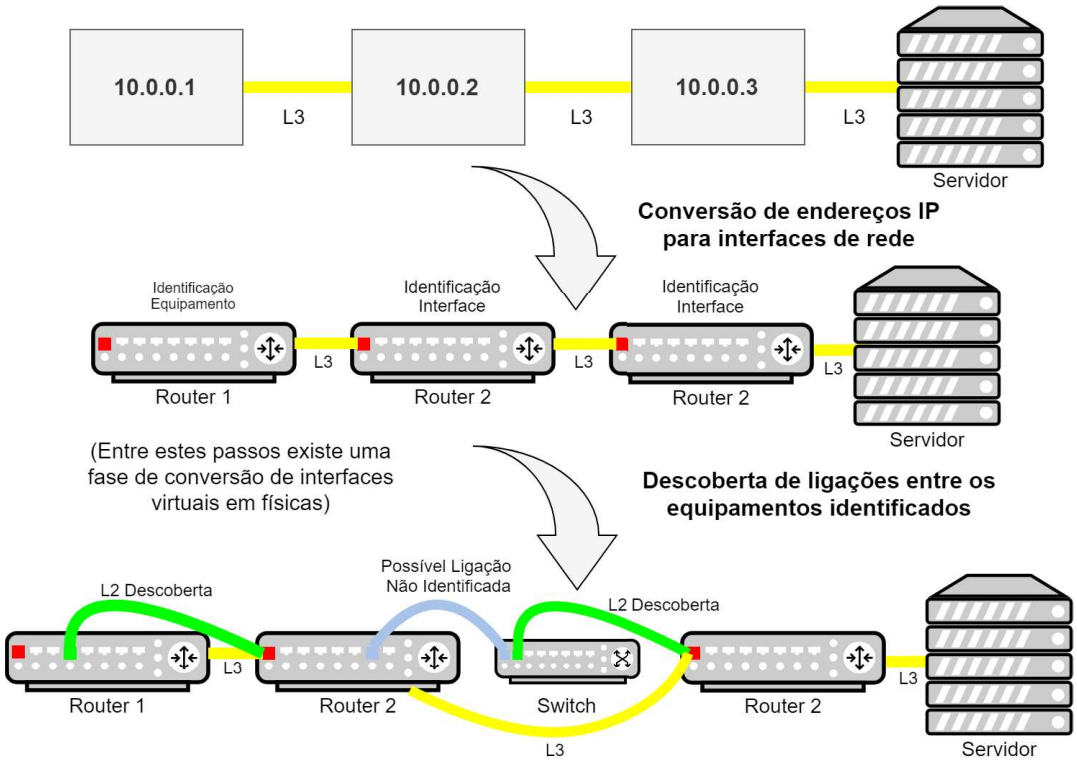


Figura 3.2: Mecanismo de descoberta da rede para cada servidor e rede indicada

fibra ótica (porta A da Figura 3.3), em dois raios de luz, cada um destes raios corresponde a uma porta do *splitter* (portas B e C da Figura 3.3). Com esta técnica é possível duplicar o tráfego de uma ligação Ethernet, permitindo assim que uma das portas usadas possa ser utilizada para fins de monitorização.

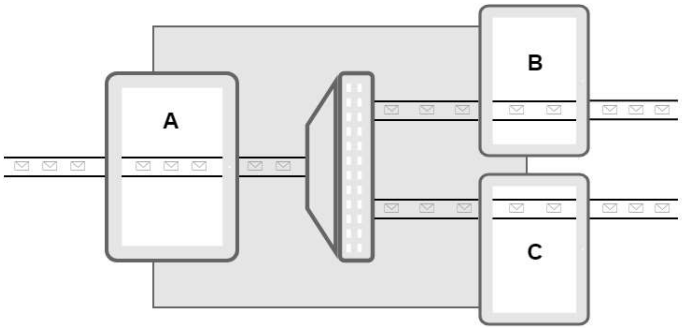


Figura 3.3: Arquitetura interna de um *splitter*

Para determinar o conjunto de pontos ótimos para a colocação de *splitters*, o **PlacementEngine**, usa um algoritmo que tem como *input* o conjunto de interfaces descobertas, com informação sobre as rotas cobertas por cada uma, e dá como *output* soluções obtidas através da aplicação de um algoritmo *greedy*. Cada solução tem um conjunto de interfaces que cumprem os requisitos mínimos de cobertura, definidos no ficheiro de configuração.

O algoritmo criado pode ser descrito, sumariamente, em vários passos:

1. Calcular o *rating* de relevância para cada interface válida.
2. Escolher a interface com o maior *rating* com uma velocidade compatível, menor ou igual à dos *splitters* e das portas dos *packet flow switches*. Adicionar a interface à solução atual.
3. Se a solução atual cumprir os requisitos mínimos é guardada como uma solução válida. Os requisitos mínimos são definidos no ficheiro de configuração em dois campos diferentes, um deles para a cobertura global mínima e outro para a cobertura global mínima de cada serviço.
4. Repetir os passos anteriores até a cobertura global ser de 100%
5. Devolver todas as soluções válidas encontradas.

O algoritmo encontra-se descrito em detalhe nos anexos A.1 e A.2.
O cálculo do *rating* de cada interface válida é feito, com base, nos seguintes critérios:

- Número de rotas que ainda não foram cobertas na iteração atual do algoritmo.
- Frequência de cada uma das rotas (número de interfaces em que a rota aparece).
- Cobertura do serviço associado ao destino de uma rota.
- Velocidade da interface, tendo em atenção o número de conexões (as agregações de rede são compostas por várias conexões).

Um outro critério que podia ter sido considerado era o volume de tráfego. O volume de tráfego define-se como a quantidade de pacotes que passam por uma interface de rede. Este critério não foi utilizado porque o indicador do volume de tráfego considera todo o tráfego como uniforme. Desta forma, não é possível verificar se o volume de tráfego esta diretamente relacionado com o tráfego relativo a um dos serviços que se pretendem analisar. Assim, não é possível verificar o tipo de tráfego específico que ali circula apenas com este indicador.

Cada *rating* é calculado, com base nos critérios descritos, da seguinte forma:

$$U = \text{rotas não cobertas da interface}$$

$$|U| \times \sum_{r \in U} \frac{1}{frequencia(r) \times cobertura\_servico(r.servico)} \times \frac{|conexoes|}{velocidade}$$

O *rating* pode ser decomposto em três componentes:

1. $|U|$ - A cardinalidade do conjunto U tem como objetivo aumentar o *rating* das interfaces com o maior número de rotas não cobertas.

2. $\sum_{r \in U} \frac{1}{frequencia(r) \times cobertura\_servico(r.servico)}$ - Esta soma tem como propósito aumentar o *rating* das interfaces com rotas menos frequentes, daí o inverso, favorecendo assim rotas com uma menor frequência. Para além da métrica associada à frequência, é dada uma maior importância às rotas de serviços com pouca cobertura, dadas as rotas já cobertas pelo algoritmo.

3. $\frac{|conexoes|}{velocidade}$ - Este componente é usado para privilegiar as interfaces com velocidade mais baixa, desta forma é possível garantir uma solução que leva em consideração a largura de banda necessária para implementar a solução de inspeção de tráfego. O número de conexões serve apenas para as agregações de rede. Este tipo de interfaces virtuais são tratadas como uma só interface, mas são compostas por várias interfaces físicas. Dessa forma não se pode cobrir uma só interface da agregação, mas sim todas as interfaces que a compõem.

A finalidade da aplicação do *rating* é a escolha da interface que cumpre melhor os critérios definidos, de uma maneira quantitativa. Pelo que, quanto mais elevado for o *rating* de uma interface maior é a probabilidade da interface pertencer a uma solução identificada pelo algoritmo.

Com a aplicação do algoritmo *greedy* e com os critérios usados não é possível garantir uma solução ótima, mas sim uma aproximação dessa solução. A solução ótima apenas pode ser encontrada através da procura exaustiva. Não existe outro algoritmo de tempo polinomial, sem ser baseado na premissa por trás do algoritmo *greedy*, que consiga dar uma melhor aproximação que resolva o problema do *minimum set cover*, a não ser que $P = NP$ [15].

Os critérios referidos servem de otimização ao algoritmo *greedy* já referido na Secção 2.3. A aplicação dos critérios de velocidade e cobertura dos serviços está exclusivamente associada ao problema da cobertura de interfaces tendo em consideração o tráfego que circula na rede. Por outro lado, os critérios da frequência de elementos e elementos não cobertos servem de otimização para o problema *minimum set cover*.

O algoritmo de procura exaustiva, ou força-bruta, seria também uma opção válida. Encontrar a solução ótima para o problema de cobertura mínima de um conjunto implica a procura exaustiva de todas as soluções. Se o universo de elementos for de grande dimensão esta opção torna-se rapidamente impraticável, mesmo com alguns melhoramentos ao algoritmo, *e.g* começar pelas combinações de conjuntos com poucos elementos e parar assim que for encontrada uma solução que garanta 100% de cobertura do universo.

A utilização de um algoritmo de força-bruta para qualquer problema tem sempre problemas de escalabilidade, como tal só deve ser utilizada se não existir mais nenhuma opção ou se o conjunto não tiver uma dimensão na ordem dos milhares/milhões de elementos. Se o algoritmo de força-bruta fosse aplicado a este problema o requisito de

escalabilidade não seria cumprido.

O universo relativo ao problema de cobertura estudado neste projeto pode rapidamente tornar-se num universo com milhares de interfaces, devido à dimensão de uma rede complexa. Desta forma a procura exaustiva é inviável para o problema que se pretende resolver.

Um outro fator que levou à escolha do algoritmo *greedy* foi a sua versatilidade. Esta versatilidade traduz-se na facilidade com que se pode modificar o algoritmo para escolher, de forma ponderada, o elemento do conjunto de interfaces válidas que cumpra melhor os critérios definidos. O algoritmo base diz que são escolhidas primeiro as interfaces que cobrem mais elementos. Essa proposição é facilmente ajustável para ser possível introduzir diferentes critérios relacionados com a cobertura de elementos/interfaces de rede, aproximando a solução encontrada da solução ótima.

Sendo o output do algoritmo um conjunto de soluções que cumprem os requisitos mínimos, é necessário encontrar a solução ótima e a solução com a melhor relação cobertura-custo. Neste caso a solução ótima é a última das soluções encontradas pelo algoritmo. A solução com a melhor relação cobertura-custo é aquela que apresenta a melhor cobertura dado o número de *splitters* da solução. Esta solução é encontrada através do cálculo do rácio $cobertura \ / \ |splitters|$ para cada elemento do conjunto de soluções. A solução com o melhor rácio é a aquela que apresenta a maior cobertura com o menor custo, *i.e* número de *splitters* necessários para implementar a solução.

3.2.3 Geração de relatórios

Ambas as soluções encontradas pelo PlacementEngine devem ser especificadas num formato apresentável para o utilizador do sistema, respeitando assim o requisito de **usabilidade**. O formato considerado foi um relatório com todos os dados necessários para implementar uma solução. Para além dos relatórios são também gerados mapas visuais da rede, de modo a ter uma perceção da estrutura física da rede.

A geração de relatórios é feita com base nos resultados da fase anterior. Por cada uma das duas soluções encontradas é gerado um relatório com os seguintes detalhes:

- Tipo, nome, velocidade, *hostname* (equipamento de rede onde está a interface) e percentagem de cobertura da totalidade das rotas de cada interface a cobrir (interfaces estão agrupadas por *datacenter*). As interfaces virtuais têm ainda uma listagem das interfaces físicas que as compõem.

- Percentagem de cobertura em relação á totalidade das rotas, que são possíveis de cobrir, associada à solução.

- Número de *splitters* necessários para cumprir os detalhes especificados.

- Totalidade de largura de banda necessária para capturar e inspecionar o tráfego de cada uma das interfaces indicadas.

Para além dos relatórios, são geradas duas imagens (com uma composição igual à Figura 3.4) representativas do mapa de rede, com a colocação física dos *splitters*, para cada uma das soluções encontradas.

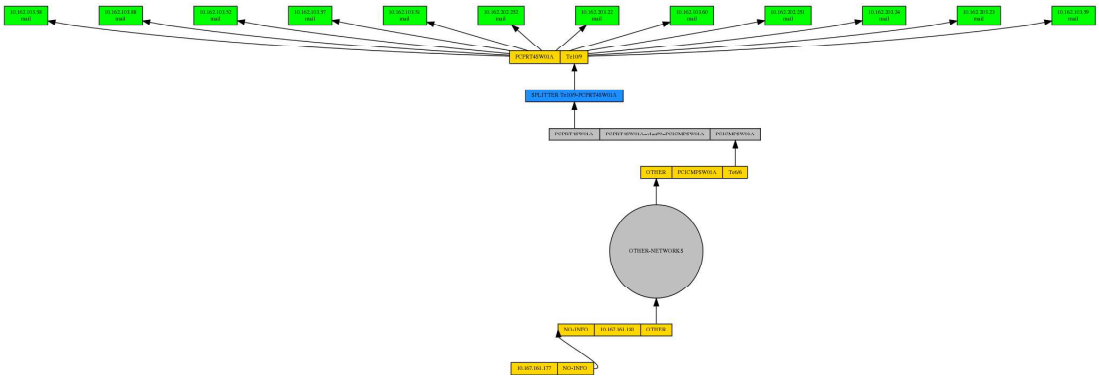


Figura 3.4: Exemplo de uma rede de rede com a colocação dos splitters, a azul

3.3 Conclusão

Este capítulo descreve a arquitetura do sistema desenvolvido e todas as decisões necessárias para chegar ao desenho final da arquitetura. Os requisitos que o sistema deve cumprir foram também explicados neste capítulo. Algumas decisões de desenho do sistema foram influenciadas por esses mesmo requisitos.

A primeira fase do sistema, a fase de descoberta, é caracterizada, abstratamente, neste capítulo. A criação da arquitetura do processo de descoberta passou por uma fase de experimentação, descrita neste capítulo, sendo a arquitetura final o resultado desta fase de experimentação. A arquitetura final do processo de descoberta é baseada no *traceroute*, tirando partido de um repositório de informação, neste caso o NNM, para poder completar a informação das interfaces recolhida através do *traceroute*.

Em relação á fase de determinação de pontos para a colocação de sensores este capítulo apresenta o algoritmo e todos os critérios tidos em consideração para a determinação desses pontos. Por fim são ainda referidos os componentes do *output* final do sistema.

O próximo capítulo especifica em detalhe a implementação de todos os componentes da arquitetura do sistema. Desde a fase de descoberta, baseada no *traceroute*, até à implementação do algoritmo de colocação de sensores descrito neste capítulo.

Capítulo 4

Implementação

No capítulo anterior foi explicada a arquitetura do NIDSPlacer, bem como cada componente do sistema, isto para que fosse possível entender a função de cada um dos três módulos do NIDSPlacer de uma forma abstrata, mas explicita.

Neste capítulo é descrita, em detalhe, a implementação do NIDSPlacer e dos seus componentes individuais.

O sistema foi implementado em Ruby, corre em sistemas operativos baseados no Linux, em que as ferramentas *ping* e *traceroute*[6] se encontrem disponíveis, e para além disso, necessita de privilégios administrativos para poder funcionar corretamente.

4.1 DiscoveryEngine

O DiscoveryEngine é o componente responsável pela descoberta da rede por onde passa o tráfego proveniente das redes indicadas no ficheiro de configuração. Este tráfego tem como destino os servidores a identificar que correm os serviços indicados pelo utilizador do sistema.

Através da observação Figura 4.1 é possível identificar todos os componentes do DiscoveryEngine. Os três componentes essenciais para o funcionamento do sistema são:

- **ResourceIdentifier -** Serve como camada de abstração para a obtenção de informação relativa a servidores e equipamentos de rede.
- **RequestCreator -** Responsável por criar um pedido por cada interface, correspondente a uma rede diferente, e servidor crítico identificado. Cada pedido é depois colocado numa fila partilhada para consumo das BuildThreads.
- **BuildThread -** Cada uma destas *threads* retira pedidos da RequestQueue e converte-os em caminhos de rede que depois são agregados num só mapa da rede. O funcionamento deste componente é explicado em mais detalhe na Subseção 4.1.3.

Estes três componentes têm implementações independentes, respeitando assim o requisito da **modularidade**.

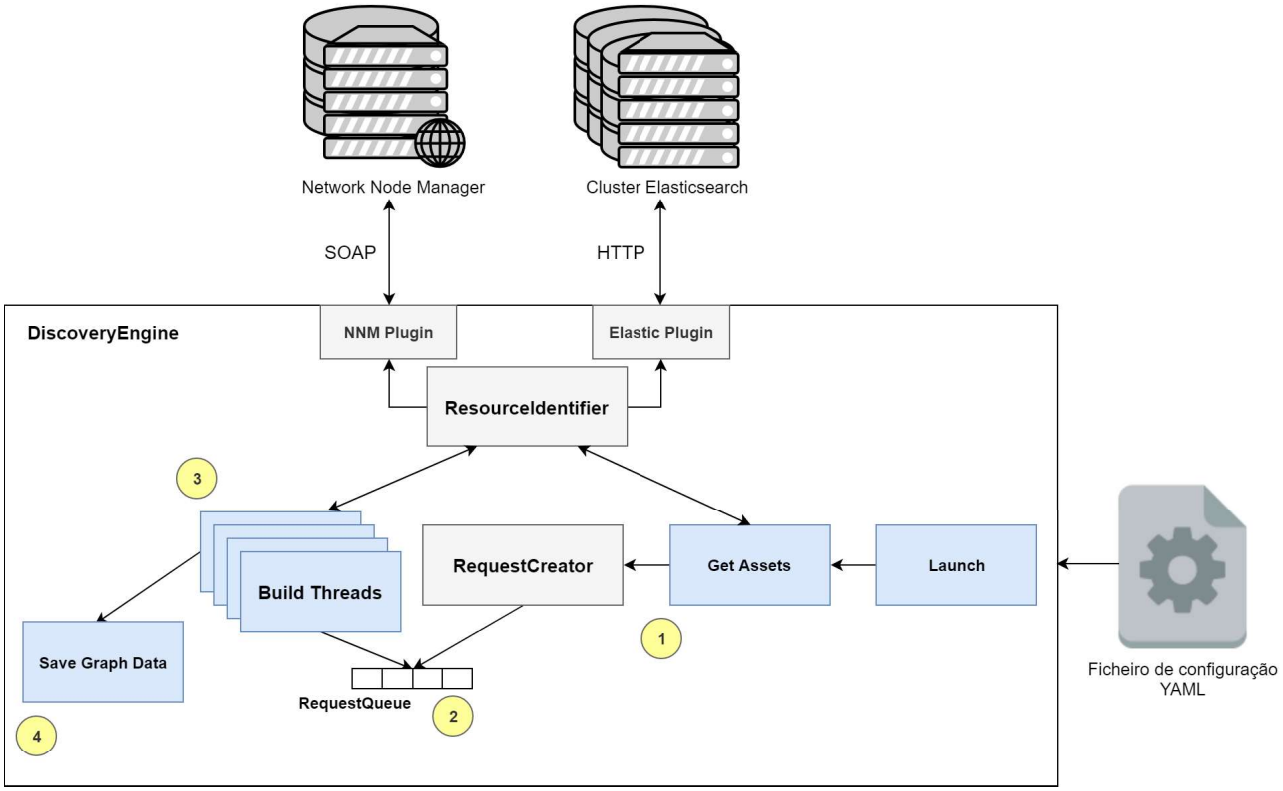


Figura 4.1: Arquitetura do DiscoveryEngine

Antes de entrar nos detalhes da implementação de cada componente, é importante explicar o processo feito pelo **DiscoveryEngine** para gerar uma representação correta da rede, na forma de um mapa.

O processo criado segue os seguintes passos:

1. Começa por pedir ao **ResourceIdentifier** os endereços IP de *frontend* (acedidos pelos utilizadores dos serviços) dos servidores que alojam os serviços indicados. O ficheiro de configuração contém campos onde devem ser colocados os serviços que se pretendem analisar.

2. De seguida, o **RequestCreator** cria vários pedidos. Um pedido por cada endereço IP e interface (indicadas no ficheiro de configuração). Cada pedido é depois colocado na RequestQueue para ser consumido por uma das BuildThreads.

3. Assim que um pedido é colocado na RequestQueue uma das **BuildThreads** vai retirar o pedido de lá, e vai iniciar o processo de descoberta do caminho de rede, com os dados que compõem o pedido.

4. Depois de todas as BuildThreads terminarem o seu trabalho, *i.e* nenhuma das *threads* estar a correr e não haver pedidos pendentes na RequestQueue, é possível gravar o mapa numa estrutura de dados propositadamente criada.

4.1.1 ResourceIdentifier

O **ResourceIdentifier** é o módulo interno responsável por obter todas a informações correspondentes aos equipamentos de rede e servidores internos.

Para obter informações sobre as interfaces e os equipamentos da rede interna da MEO, foi desenvolvido um *plugin*, o **NNMPlugin**, de modo a auxiliar o processo de descoberta baseado no *traceroute*. Este sub-componente é responsável por invocar a API do NNM, a plataforma interna de gestão de equipamentos de rede da MEO, para a obtenção de metadados associados aos equipamentos de rede.

A informação relativa aos servidores internos da MEO encontra-se armazenada num índice dum *cluster* Elasticsearch. Para obter essa informação, é necessário contactar o *cluster* via HTTP através de uma API RESTful para obter o conteúdo filtrado do índice. O sub-componente responsável por essa tarefa é o **ElasticPlugin**.

Os sub-componentes por trás do ResourceIdentifier foram criados a pensar, especificamente, nas plataformas internas já existentes na MEO. A implementação dos sub-componentes, noutra organização, deve ter como base as plataformas internas que armazenam os metadados respeitantes a equipamentos de rede e servidores. A implementação destes componentes não precisa de estar necessariamente associada à obtenção de dados das plataformas internas. É possível obter os mesmos dados através de pedidos feitos diretamente aos equipamentos de rede ou servidores. No caso de já existirem plataformas com esta informação é sempre preferível a sua utilização, de modo a não adicionar carga desnecessária aos equipamentos.

Para este componente e os seus sub-componentes funcionarem é necessário configurar o sistema de maneira correta. Para isso acontecer, existem vários campos no ficheiro de configuração YAML, **config.yml**, para esse efeito. Para configurar o ResourceIdentifier são necessários os seguintes campos:

- **Datacenters** - Acrónimos, com 3 letras, que representam os *datacenters* físicos em que devem ser identificados os servidores pertencentes aos serviços indicados - **target_assets: locations**, *e.g* dc-pic.

- **Tipos de serviços a analisar** - A chave dum campo deste tipo deve ser o tipo do serviço e o valor do campo deve ser composto por uma palavra-chave que todos os servidores daquele tipo tenham no seu nome, e ainda a cobertura mínima para este tipo de serviço - **Campos colocados debaixo de index_assets**.

- **Redes a mapear** - Nome das redes que devem ser mapeadas, de acordo com a informação do NNM e dos outros cadastros de rede da MEO - **config: networks_to_map**

Os dois *plugins* podem ser configurados preenchendo corretamente os seguintes campos:

- ElasticPlugin
  - Índice do repositório de equipamentos internos (servidores, equipamentos de rede e *endpoints*) - **target_assets: index_assets**
  - Índice do repositório de equipamentos de rede internos (este índice contém informação mais completa e detalhada relativamente aos equipamentos de rede) - **config: index_network_info**
  - Credenciais de acesso ao *cluster* e *timeout* de uma invocação, em segundos - **server**
- NNMPlugin
  - Credenciais de acesso à API - **service_account**
  - Número de *retries* a fazer numa invocação da API e o *timeout* da invocação, em segundos - **nnm_query**

Como este componente serve de abstração para a implementação dos sub-componentes que permitem a obtenção de informação, a sua implementação apenas aponta para as funções implementadas pelos *plugins*. Como tal, a complexidade da implementação está nos *plugins* e não na camada de abstração em si. Todas as funções descritas seguidamente são apenas invocadas através do ResourceIdentifier e não diretamente aos *plugins*.

ElasticPlugin

Este *plugin* é responsável por fornecer uma API com funções que possibilitem a extração de dados de um *cluster* elastic. Como o âmbito deste projeto está relacionado com servidores e equipamentos de rede a API fornecida permite a obtenção de informação relativa a ambos.

A comunicação com o *cluster* é feita através de HTTP, utilizando um URL que permite o acesso ao *cluster*. O URL é composto pelo conjunto de parâmetros associados à chave **server** do ficheiro de configuração (***host*** - URL do *cluster*, ***port*** - porta de acesso, ***user*** - utilizador, ***pass*** - password, ***proto*** - protocolo, ***timeout***). O cliente que permite a conexão com o *cluster*, via HTTP, é uma *gem* Ruby disponibilizada em [12].

A API exposta é composta pelas funções:

- **get_assets** - Esta função obtém todos os endereços IP *frontend* dos servidores que alojam um dos tipos de serviços indicados no ficheiro de configuração. Para isso acontecer é feita uma *query* ao índice especificado no campo **index_assets**.

  A *query* filtra os servidores por plataforma e localização física. As plataformas são identificadas pela palavra-chave em cada tipo de serviço no ficheiro de configuração. As localizações, também definidas no ficheiro de configuração, estão especificadas

no campo **locations** e são utilizadas em diversos componentes do sistema. O resultado desta função são todos os endereços IP *frontend* resultantes da *query*.

- **get_location -** O propósito desta função é devolver o *datacenter* em que está um certo equipamento. O índice aqui utilizado deve ser aquele que tem a informação de localização estandardizada ou estruturada de forma semelhante. Sem essa garantia não é possível garantir a consistência dos dados obtidos sobre a localização dos equipamentos. O índice que contém esta informação estandardizada corresponde, neste caso, ao campo **index_assets**. O único parâmetro desta função é o *hostname*/IP de gestão do equipamento. No caso de ser um endereço IP válido a *query* é feita com um campo específico para IPs, caso contrário a *query* é feita com um campo para *hostnames*.

  A *query* retorna a localização do equipamento indicado como parâmetro. A localização é depois convertida para um acrónimo com três letras, de modo a ficar equivalente aos acrónimos de *datacenter* do ficheiro de configuração.

- **is_equipment_in_network -** Esta função verifica se um equipamento de rede pertence a uma rede que deve ser mapeada. O índice utilizado deve ser aquele que tem a informação de equipamentos de rede mais completa, *e.g* um índice que cruze informação de diferentes cadastros de rede. O índice ao qual é feita a *query* deve estar especificado no campo **index_network_info**. A implementação deste método está dependente da estrutura da informação de uma plataforma interna da MEO, o IP Address Management (IPAM). Devido a essa dependência, a *query* só funciona para o formato de informação do IPAM. Esta função poderia ter sido implementada no NNMPlugin, mas o cruzamento de cadastros (IPAM e outros cadastros internos) de rede internos num só índice do elastic fez com que fosse implementada como uma função do ElasticPlugin.

  A *query* indica qual o endereço IP, pode ou não ser o IP de gestão do equipamento. O resultado proveniente da invocação da *query* retorna um campo que contém as regiões/redes em que o equipamento se inclui. Para verificar se o equipamento se inclui nas redes a mapear é feito um cruzamento entre a informação obtida e o campo **networks_to_map**, se houver correspondência o equipamento encontra-se numa rede que deve ser mapeada.

As *queries* descritas nas funções são feitas com invocações da API elastic, através do método *search*. Este método contacta o *cluster*, de modo a correr uma *query* sobre o índice indicado como parâmetro. A comunicação é feita via HTTP. Se ocorrer uma falha este processo é repetido até ser atingido o número de *retries* máximo indicado em **max_retries**.

Os métodos get_location e is_equipment_in_network usam uma *cache*, implementada no ResourceIdentifier, para evitar fazer *queries* ao *cluster* que já foram feitas anteriormente, sendo assim possível responder mais rapidamente a uma invocação de um destes métodos. O acesso é essa *cache* é controlado por *locks*. Assim estes métodos podem ser invocados por várias *threads* concorrentemente, sendo por isso necessário um mecanismo que controle a concorrência.

Todas as *queries* sobre um índice elastic são feitas com a linguagem Lucene [17].

NNMPlugin

A plataforma interna de gestão de equipamentos de rede da MEO é o NNM. Como tal, o *plugin* que obtém toda a informação relativa aos equipamentos de rede foi baseado nas funções que compõem a API SOAP do NNM [13].

A implementação deste *plugin* é relativamente simples, todas as funções implementadas tratam de invocar a API com os parâmetros corretos e devolvem a informação estruturada numa *hash* para uma manipulação dos dados mais simples. A implementação é *thread-safe* podendo todos os métodos ser acedidos por várias *threads* concorrentemente.

Para invocar os diferentes serviços fornecidos pela API via SOAP, é necessário um cliente por cada tipo de serviço. Foram criados seis clientes: **NodeClient, IPAddressClient, InterfaceClient, SNMPClient, VLANClient, NodeGroupClient**. Cada cliente é inicializado com as credenciais de acesso especificadas no ficheiro de configuração, em **service_account**. Para além das credenciais, cada cliente tem a ele associado uma interface WSDL do serviço web, uma *cache* para responder aos pedidos mais rapidamente e uma *lock* para controlar o acesso concorrente aos recursos disponibilizados pelo *plugin*. Essa interface é definida num ficheiro WSDL disponibilizado pelo NNM. Cada função implementada por cada um dos clientes usa a biblioteca SOAP Savon [1] para fazer a comunicação com a API NNM.

Este *plugin* fornece as seguintes funções:

- **translate_ip_to_hostname**

  Traduz um IP para o nome do equipamento de rede associado ao IP.

- **get_node_info**

  Devolve toda a informação sobre um equipamento (data de criação, descrição, categoria, id, nome, localização física e nome de sistema).

- **is_equipment_in_network**

  Verifica se um equipamento está nas redes a mapear.

- **get_interface_info**

  Devolve todas as informações associadas a uma interface (descrição, índice, nome, id, velocidade em gigabits por segundo - Gbps).

- **get_interface**

  Converte uma interface virtual (VLAN ou agregação) em interfaces físicas.

- **get_if_name_from_index**

  Devolve o nome da interface com um determinado índice num equipamento de rede.

- **is_aggregation**

  Verifica se uma interface corresponde a uma agregação de rede.

- **belongs_to_aggregation**

  Verifica se uma interface pertence a uma agregação de rede.

- **get_device_l2_connections**

  Devolve todas as conexões de L2 de um equipamento.

- **get_ip_info**

  Retorna todas as informações associadas a um determinado IP (id do equipamento a que pertence, id da interface, id da sub-rede e IP).

- **get_interface_vlan**

  Retorna a interface física por onde passa o tráfego, de uma VLAN, vindo de um equipamento de rede.

4.1.2 RequestCreator

O **RequestCreator** é o módulo do DiscoveryEngine responsável por criar e colocar pedidos na RequestQueue. Os pedidos criados indicam os caminhos de rede a descobrir pelas BuildThreads.

Antes deste módulo ser inicializado é necessário alterar algumas configurações do *kernel* para que seja possível receber pacotes de diferentes interfaces. O sistema trata de mudar as configurações e de revertê-las assim que termina, sendo dessa forma preciso privilégios administrativos da máquina. As configurações modificadas são referentes ao *reverse path filtering*[1] feito, pelo *kernel* do Linux, para cada interface da máquina. Cada interface a utilizar pelo sistema deve portanto ter esta funcionalidade desativada para que possa funcionar corretamente. A mudança de configuração é necessária porque os clientes de VPN utilizados na MEO (qualquer outro cliente terá provavelmente o mesmo comportamento) criam entradas na tabela de encaminhamento que indicam que a grande maioria do tráfego deve passar pela interface associada à VPN, impedindo o tráfego de ser encaminhado para outras interfaces sem alterar as configurações de encaminhamento manualmente.

Este facto faz com que as respostas aos pacotes ICMP que tenham sidos enviados por uma interface que não a da VPN sejam descartadas por serem provenientes de uma interface pela qual o pacote não seria encaminhado, isto porque as rotas da VPN se sobrepõem as outras por serem mais especificas, a única exceção são os pacotes que circulam na rede local de cada uma das interfaces.

[1] Se a resposta de um pacote vier por uma interface pela qual não seria encaminhado o pacote e a resposta

Assim que o *reverse path filtering* é desativado para cada uma das interfaces a usar pelo sistema o processo do RequestCreator pode ser inicializado. O processo é bastante simples, na medida em que apenas delega outros processos, neste caso as BuildThreads, para realizar os pedidos, tratar as respostas e construir o mapa da rede. Podemos afirmar que este processo é o produtor e as BuildThreads as consumidoras, parafraseando o problema do produtor-consumidor. O *buffer* partilhado entre eles, a RequestQueue, foi implementado com uma Queue *thread-safe* nativa do Ruby [10].

A inicialização deste processo é feita assim que são identificados os servidores que alojam os serviços pertinentes ((1) da Figura 4.1). Por cada servidor e interface, indicada no ficheiro de configuração no campo **network_interfaces**, é criado um pedido com três componentes:

1. **Endereço IP de *frontend* do servidor**;
2. **Uma das interfaces da máquina onde corre o sistema**;
3. **A que rede corresponde a interface** (VPN, Wi-Fi, uma rede interna por cabo, etc.);

Os pedidos criados servem para a descoberta dos caminhos entre todas as redes indicadas e todos os servidores, criando assim o mapa da rede.

Depois de criado, cada pedido é adicionado à RequestQueue para ser consumido por uma BuildThread ((2) da Figura 4.1).

4.1.3 BuildThreads

As BuildThreads são talvez o componente mais complexo deste sistema, isto porque estas *threads* implementam o processo que permite converter um conjunto de elementos da rede, as redes internas e os servidores, num mapa de rede que represente toda a infraestrutura de rede descoberta entre esses elementos.

As *threads* são colocadas numa *thread pool*, de modo a serem geridas de maneira mais eficiente. A *pool* termina assim que todas as *threads* tiverem terminado o seu trabalho. O número máximo de *threads* na *pool* é dado pelo parâmetro **num_threads** do ficheiro de configuração. Se não for colocado nenhum valor neste parâmetro o número de *threads* vai ser igual ao número de *cores* da máquina local. A utilização de *threads* serve para tirar partido da arquitetura *multi-thread* dos sistemas mais modernos.

Cada uma das BuildThreads tenta retirar um pedido da RequestQueue para o processar. Se não houverem pedidos pendentes na fila a *thread* dá o seu trabalho como terminado. Quando retira um pedido da RequestQueue o processo de descoberta, ilustrado na

são descartados pelo *kernel*[5]. Esta técnica é usada para evitar o *spoofing* de endereços IP.

Figura 4.2, é inicializado. Este processo corre sempre que é processado um novo pedido por uma das *threads*.

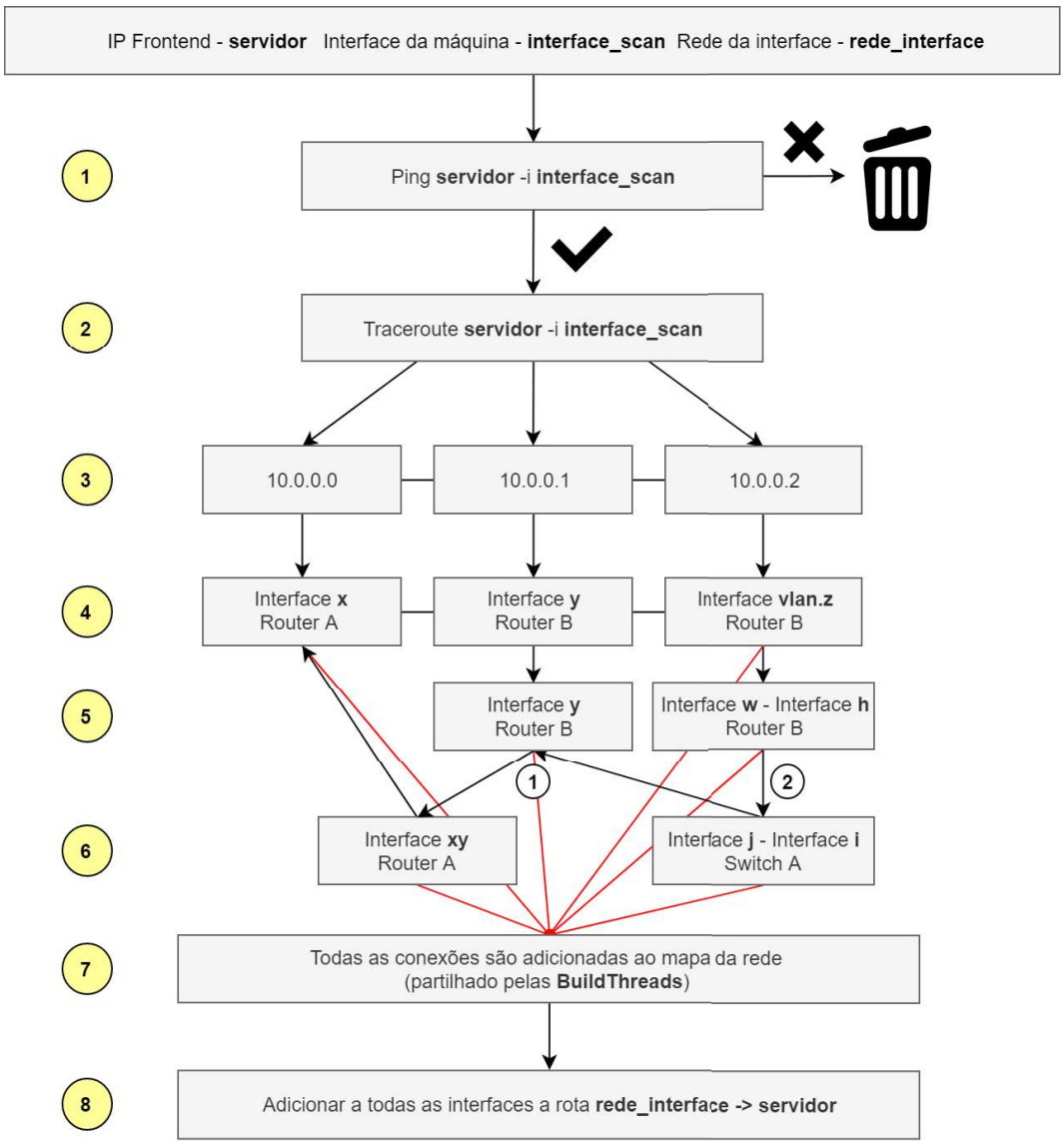


Figura 4.2: Processo de descoberta do caminho de rede

Com os dados de um pedido, **servidor**, **interface_scan** e **rede_interface**, é possível começar o processo de descoberta. O processo segue os seguintes passos sequenciais:

① **Verificação da conectividade** entre a **interface_scan** e o **servidor**

Esta verificação é feita com a ferramenta *ping*, disponível em qualquer distribuição Linux. O *ping* verifica a conectividade, através de ICMP, entre uma interface da máquina de onde é executado e um endereço IP. O campo **ping_retries**, do ficheiro de configuração, indica o número de vezes que deve ser feito o *ping*. Só é considerado que não há conectividade com um servidor quando 100% dos pacotes ICMP

são perdidos e não há uma resposta válida. No caso de não haver conectividade o pedido é descartado e a *thread* tenta retirar outro pedido da RequestQueue.

(2) **Obtenção de todos os equipamentos de L3** desde o *default-gateway*, da rede associada à **interface_scan**, até ao **servidor** indicado

A lista dos equipamentos de L3 é representada por todos os endereços IP que decrementaram o TTL dos pedidos ICMP. A ferramenta **traceroute**, normalmente disponível em qualquer distribuição Linux, foi utilizada para obter esta listagem. A implementação do *traceroute* com ICMP foi escolhida por ser a que tem o melhor desempenho face à sua simplicidade e por haver garantia que os pedidos ICMP não são bloqueados por uma *firewall*, isto porque houve uma verificação de conectividade no passo anterior. Para ser possível fazer um *traceroute* através de ICMP são necessários privilégios elevados.

(3) Os elementos do *tracerout* são agrupados em pares. No caso de (3) da Figura 4.2 **- 10.0.0.1→10.0.0.0** e **10.0.0.2→10.0.0.1**. Os elementos são agrupados em pares para serem descobertas as ligações entre eles, com exceção do primeiro elemento. Os pares são feitos entre todos os *hops* do *traceroute* e o *hop* anterior associado a cada um deles. Cada um dos endereços do *traceroute* representa a interface de entrada do tráfego num determinado equipamento. Para se fazerem as conexões é necessário descobrir a(s) interface(s) de saída do equipamento anterior, quer ele seja o *router* agrupado com ele (*hop* anterior), um *switch* ou um equipamento em *bridge*. A partir deste passo a ordem seguida por todas as operações é a ordem do *traceroute*, *i.e* 10.0.0.1→10.0.0.0 e 10.0.0.2→10.0.0.1.

(4) **Cada endereço IP é convertido numa interface** com a função **get_ip_info** do ResourceIdentifier. Com os dados obtidos com a invocação do método é possível obter o *hostname* associado à interface, através do método **get_node_info**.

As interfaces devolvidas pelo get_ip_info podem ser de quatro tipos:

- Interface física (*e.g* Te1/2)
- Interface física com VLAN ID (*e.g* Te1/2.**99** - VLAN ID 99)
- VLAN (*e.g* Vl99, vlan.99, VLAN-99, etc.)
- Agregações de rede (*e.g* Po70, Bundle-Ether70, etc.)

(5) **Cada interface identificada no passo anterior passa por um mecanismo de conversão**, **get_interface** do ResourceIdentifier, para converter interfaces virtuais em físicas. O equipamento atual, primeiro equipamento do par, e o anterior, segundo do par, são utilizados neste passo para encontrar a interface correta do tráfego proveniente do equipamento anterior do *traceroute*, e segundo elemento do par.

A conversão é feita seguindo três passos:

1. As interfaces físicas não passam por este processo. Uma interface física, com ou sem VLAN ID, é identificada pelo seu nome e não precisa de ser convertida.
2. É obtido o índice da interface virtual associado ao equipamento de rede identificado.
3. A interface é convertida de maneira diferente para uma VLAN e para uma agregação de rede.
   - Se for uma VLAN (as interfaces físicas com VLAN ID não passam por este processo) é invocada a função **get_interface_vlan** que devolve a interface física por onde passa o tráfego, associado à VLAN, vindo do equipamento agrupado para o atual. A função obtém esta informação através da obtenção da tabela CAM (tabela de encaminhamento L2) do equipamento atual.
     A tabela é obtida, se não estiver em *cache*, com um pedido SNMP, via API do NNM, com o OID 1.3.6.1.2.1.17.4.3 sendo que existem subtabelas para cada VLAN. Cada entrada da tabela é tratada, para que cada MAC identificado possa corresponder à interface por onde passam os pacotes com origem nesse MAC. Depois de obter a tabela CAM associada à VLAN é necessário identificar a interface por onde passa o tráfego proveniente do equipamento anterior. Para encontrar essa interface basta encontrar a entrada que corresponde ao único MAC que coincide com uma interface do equipamento anterior, ou a única interface do equipamento anterior que está associada à mesma VLAN. A interface que corresponder ao MAC é a que faz a conexão com um equipamento anterior ao atual, que pode ou não ser o anterior, visto que pode haver vários *switches* entre dois equipamentos de L3. É ainda necessário verificar se a interface é ou não uma agregação, e em caso afirmativo, é preciso convertê-la.
   - No caso de ser uma agregação de rede é necessário verificar se a interface indicada é de facto uma agregação. Para isso é necessário extrair toda a tabela de agregações do equipamento, com um pedido SNMP, feito através da API do NNM, que por sua vez vai contactar o equipamento via SNMP. Esta é uma das poucas operações que implica um contacto direto ao equipamento, já que esta informação não se encontra disponível no NNM. O método **is_aggregation** trata de extrair a informação das agregações, se necessário, porque a tabela pode já estar em *cache*, e verifica se o índice da interface pertence à tabela.
     A tabela das agregações é a junção de uma ou duas tabelas. Uma é a

tabela do Link Aggregation Control que é extraída via SNMP, com o OID 1.2.840.10006.300.43.1.2.1. No caso do equipamento ser Cisco, é também extraída a tabela PAgP (Port Aggregation Protocol) com o OID 1.3.6.1.4.1.9.9.98.1.1.1, isto porque esta tabela pode conter informação não representada na outra tabela.
A informação presente na tabela permite obter os índices das interfaces físicas que compõem uma agregação.

Depois de converter as interfaces físicas para virtuais, é necessário verificar se as interfaces físicas encontradas pertencem ou não a uma agregação (esta verificação não é feita para agregações que foram convertidas em interfaces físicas). A função **belongs_to_aggregation** vai verificar se cada uma das interfaces encontradas pertence a uma das interfaces que compõe uma agregação de rede do equipamento.

Esta verificação e respetiva conversão é obrigatória porque as interfaces de uma agregação devem ser tratadas como uma só interface, isto porque o tráfego que circula numa agregação circula em todas as interfaces. Isto pode não acontecer na prática porque existem portas redundantes, mas estas também devem ser levadas em consideração pois a qualquer momento podem ser ativadas.

No final do processo de transformação, as interfaces identificadas vão ser apenas agregações e interfaces físicas individuais, que podem corresponder ou não a VLANs.

(6) **Descoberta das ligações de L2** entre os equipamentos, caso existam

Os elementos do *traceroute* são agrupados aos pares para possibilitar a descoberta das ligações L2 feita neste passo.

Cada elemento do conjunto de interfaces proveniente do passo (5), pertencentes ao primeiro elemento do par, passa por um mecanismo de descoberta de conexões L2.

As conexões identificadas podem ser de vários tipos, ilustrados nas figuras 4.3 a 4.9.

Figura 4.3: VLAN numa ligação entre duas interfaces físicas

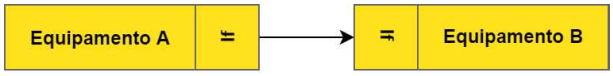


Figura 4.4: Ligação entre duas interfaces físicas

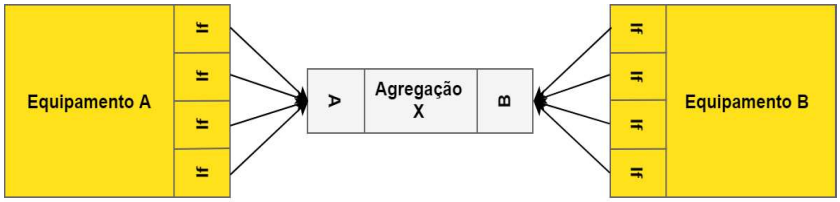


Figura 4.5: Agregação de rede composta por quatro interfaces físicas em cada equipamento

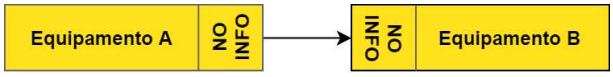


Figura 4.6: Ligação não identificada entre dois equipamentos

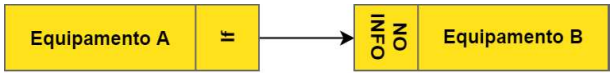


Figura 4.7: Ligação não identificada entre dois equipamentos, sendo possível identificar uma das interfaces

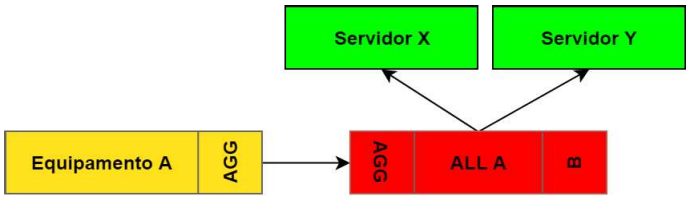


Figura 4.8: Agregador de ligações (a vermelho) criado quando não há interfaces para cobrir numa rota

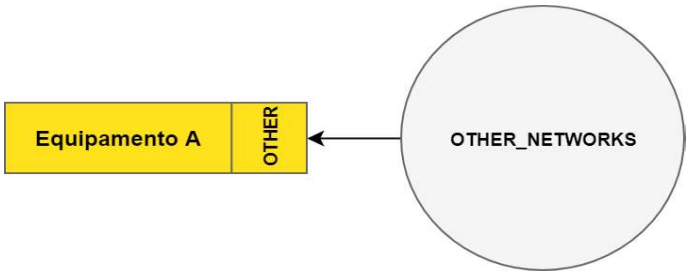


Figura 4.9: Ligação entre um equipamento e uma rede que não é mapeada

Apenas as interfaces consideradas como válidas para a solução final devem passar pelo mecanismo de descoberta das ligações L2. As interfaces inválidas são as seguintes:

- Interface não identificada devido à falta de mapeamento do equipamento anterior ou falta de mapeamento do equipamento atual (Figura 4.6)
- Ligação não identificada composta por uma interface, não identificada, de um equipamento mapeado (Figura 4.7)
- Interface correspondente à interface de interligação com uma rede que não deve ser mapeada (Figura 4.9)

Se a interface associada ao equipamento atual for uma agregação de rede é necessário criar um nó virtual que faça a ponte entre as ligações identificadas, *i.e* todas as interfaces do equipamento atual são agrupadas em conjunto com todas as interfaces do outro equipamento das ligações descobertas. Se a interface for de uma conexão a uma rede que não é suposto mapear, é criada uma conexão entre a interface **OTHER** e um nó virtual **OTHER_NETWORKS**, que representa as conexões com redes não mapeadas.

Para cada interface válida do conjunto de interfaces, é invocada a função auxiliar **get_matching_conn** de cada BuildThread. Esta função começa por procurar uma possível conexão, previamente descoberta, associada aquela interface e equipamento. Cada conexão L2 descoberta é guardada numa *cache* que cada BuildThread tem para esse propósito. No caso de haver uma *cache miss*, *i.e* a interface e equipamento ainda não têm uma ligação correspondente, é necessário obter todas as ligações de L2 do equipamento de rede através do ResourceIdentifier.

O método **get_device_l2_connections** devolve todas as ligações L2 descobertas pelo NNM através do LLDP, CDP e outros protocolos de descoberta L2. A procura pela ligação associada à interface e equipamento deve acontecer dos dois lados de cada ligação, *i.e* o par $equipamento[interface]$ deve ser procurado tanto como fonte ou destino de uma ligação descoberta, isto acontece porque o NNM agrupa todas as conexões L2 de um equipamento mesmo que sejam descobertas por outro equipamento.

No caso de não haver nenhuma ligação correspondente à interface e equipamento é criada uma conexão igual à da Figura 4.7. No caso de ter sido identificada uma ligação L2, é necessário verificar se o outro equipamento da ligação corresponde ou não ao equipamento agrupado, *i.e* ao equipamento anterior do *traceroute*.

Se houver correspondência entre os dois equipamentos do par, é feita a ligação entre os dois equipamentos. Se não houver, significa que o outro equipamento da ligação

descoberta corresponde a um *switch* (os *switches* não aparecem no *traceroute* por serem equipamentos de L2) ou a um equipamento de rede em *bridge* (*switch*, *router* ou *firewall*) que não decrementa o TTL. Quando não existe correspondência são feitas duas ligações. Uma ligação entre o equipamento atual e o equipamento descoberto, através das interfaces identificadas. A segunda ligação é feita entre o equipamento descoberto e o equipamento agrupado, sem que seja possível obter informação correta sobre a ligação entre os dois.

Não é possível obter informação correta da ligação entre dois equipamentos, em que um deles não pertence aos equipamentos do *traceroute*, apenas com o nome do *switch* ou equipamentos em *bridge*, porque as portas e equipamentos por onde circula o tráfego de/para servidor não são facilmente identificáveis apenas com essa informação. Seria possível fazer uma descoberta completa com ferramentas de descoberta de rede ou até mesmo com o desenvolvimento de um processo capaz de fazer isso, mas ambas as opções implicavam uma complexidade acrescida ao sistema e a todos os componentes já desenvolvidos. Uma exemplo de um processo a desenvolver capaz de fazer a descoberta dessas ligações poderia ser baseado na obtenção da tabela ARP, se o equipamento fosse L3, mas alguns equipamentos de rede impõem restrições sobre a obtenção destas tabelas [2] (*e.g routers* Cisco ASR), o que faria com que o processo de descoberta se tornasse bastante complexo, por ser preciso uma análise das tabelas. A informação das ligações descobertas é suficiente para implementar uma solução de monitorização suficientemente completa e correta entre os dois elementos do par, apesar de não ter sido identificada uma ligação direta entre os dois. Desta forma só é possível identificar equipamentos diretamente conectados a cada um dos equipamentos do *traceroute*.

⑦ **Adição das conexões identificadas no passo ⑥ ao mapa da rede** partilhado por todas as BuildThreads

O mapa de rede foi implementado como um grafo direcionado, *i.e* as arestas são unidirecionais, representando assim o fluxo de tráfego proveniente das redes internas para os servidores identificados. O grafo direcionado foi implementado com uma *hash* Ruby (equivalente a uma tabela *hash*). Cada nó do grafo é, unicamente identificado pelo seu nome, dessa forma é possível armazenar toda a informação relativa a um nó na entrada da tabela referente a ele. Cada entrada é composta por uma lista ligada, o primeiro elemento da lista é uma estrutura de dados que representa o nó em si e o resto dos elementos são todas as ligações que nó tem.

Um nó é composto por cinco atributos:

- **Nome**;
- **Localização física** (datacenter-x, datacenter-y, etc.);

- **Cobertura** - *hash* composta pelas interfaces do nó, sendo que cada interface tem depois uma *hash* que guarda todas as rotas que passam por ela;
- **Conexões -** Número de conexões com outros nós;
- **Id -** Id das interfaces;

Como o grafo pode ter uma grande dimensão devido ao número de equipamentos identificados, é importante salientar que as operações de criação e leitura de informação dum nó têm uma complexidade temporal constante. A procura de uma ligação é o único método com complexidade linear consoante o número de ligações de um nó.

```
  def add_connection(node1, node2, output_interface, input_interface, speed)
    if node1.nil? || !(node1.is_a? Node) || node2.nil? ||
        !(node2.is_a? Node)
      raise 'Node can\'t be nil or of the wrong type'
    end
    added = 0

    unless has_connection(node1, node2, output_interface, input_interface)

      # if node doesnt exist, as source
      added += add_node node1 unless include node1

      inc_connection node1

      # if node doesnt exist, as destination
      added += add_node node2 unless include node2

      inc_connection node2

      added = -1 if added == 0

      @lock.synchronize do
        # last argument should say if an interface is part of a vlan/aggregation group
        # connections should be reserverd because the discovery is done from last to first
        @nodes_ref[node1.name].first.add_interface_speed(output_interface, speed, node2.name.include?
('=')) @nodes_ref[node1.name].push Edge.new(node2.name, node1.name, input_interface, output_interface,
                                            speed)
        @nodes_ref[node2.name].first.add_interface_speed(input_interface, speed, node1.name.include?('='))
        @nodes_ref[node2.name].push Edge.new(node2.name, node1.name, input_interface, output_interface,
                                            speed)
      end

      puts "Added connection: #{node2.name}=#{input_interface} → " \
                     "#{output_interface}=#{node1.name} #{"with #{speed}" if speed.strip != ''}"
    end

    added
  end
```

Figura 4.10: Adição de uma conexão ao grafo

A Figura 4.10 descreve a implementação da função de adição de uma ligação ao grafo. O processo é relativamente simples. Primeiro é necessário verificar se a conexão já existe. Se não existir são criados os dois nós, caso não existam, e são

adicionadas as interfaces aos respetivos nós, o *node1* corresponde ao primeiro argumento da função e deve ser o primeiro elemento de cada uma das ligações identificadas no passo (6), o *node2* corresponde ao nó descoberto, a *outut_interface* é a interface do *node1* e a *input_interface* do *node2*. A ligação adicionada, a ambos os nós, corresponde ao padrão de tráfego real direcionado para os servidores, para isso a ligação é invertida porque a descoberta é feita na direção inversa à do tráfego que se pretende identificar, a *input_interface* passa a ser a *output_interface* e vice-versa. As implementações das operações de adição de um nó e procura de uma conexão entre dois nós são ambas *thread-safe*.

As ligações entre dois equipamentos que representem uma agregação ou VLAN implicam a adição de um nó virtual que represente essa característica. Esse nó vai ter o nome de **Equipamento A=VLAN/Agregação=Equipamento B**. Deste modo é possível garantir que não há sobreposição de nós virtuais. Cada nó virtual destes vai ter uma interface correspondente ao Equipamento A e outra ao Equipamento B, tal como se pode nas figuras 4.3 e 4.5. As conexões que correspondam a uma agregação ou VLAN são feitas do Equipamento A para o nó virtual e do nó virtual para o Equipamento B.

(8) A rota **rede_interface** $\rightarrow$ **servidor** é adicionada a todas as interfaces identificadas.

Depois de criar e adicionar todas as ligações ao mapa da rede é preciso adicionar a rota, associada à rede da interface da máquina local, que tem como destino o **servidor**.

As rotas são colocadas adicionando a rota à *hash* cobertura do nó no campo associado à interface. No caso da interface pertencer a uma agregação as rotas são adicionadas à interface do nó virtual que representa aquele equipamento, e não aos equipamentos de rede em si.

Cada processo individual de descoberta dos caminhos de rede vai gerar um conjunto de mensagens que indica a operação atual que cada uma das *threads* está a fazer naquele instante. Este conjunto de mensagens serve como *log* do processo de descoberta, podendo o utilizador do sistema analisar facilmente o conteúdo deste *log*.

Depois de todas as BuildThreads terem repetido este processo para cada pedido colocado na RequestQueue, é necessário filtrar a informação encontrada, de modo a colocar a informação mais pertinente para o PlacementEngine num formato mais indicado. O grafo que representa o mapa de rede tem como propósito a criação e geração de uma imagem que represente o mapa da rede. O PlacementEngine precisa que a rede seja organizada por interfaces, neste caso em forma de lista, em vez de organizar a rede em forma de um grafo.

É gerada uma imagem que representa o mapa de rede com o Graphviz, um *software* de visualização de grafos *open-source* [4]. O mapa de rede gerado é equivalente ao do Anexo

B.1. Os nós a amarelo representam equipamentos de rede, os nós a verde são servidores, os nós a cinzento são virtuais (VLANs, agregações de rede ou redes não mapeadas) e os nós a vermelho, Figura 4.8, são nós virtuais criados para a agregação de conexões em caminhos de rede sem qualquer equipamento ou interface válida até ao último endereço do *traceroute*. Esta distinção é útil para redes de grande dimensão por permitir distinguir facilmente os componentes da rede.

Para cada um dos nós do grafo, incluindo nós virtuais com exceção das redes não mapeadas, são identificadas todas as interfaces válidas do nó e são colocadas na lista de todas as interfaces da rede (**save_graph_data**, na Figura 4.1). As interfaces NO-INFO e OTHER não são consideradas como válidas. Se uma interface pertencer a uma agregação ela não é considerada como válida porque é necessário cobrir todas as interfaces da agregação e não apenas aquela, isto para monitorizar corretamente o tráfego que por ali passa. Para cada equipamento de rede é verificada a existência de *packet flow switches* no *datacenter* em que está, se não existirem *packet flow switches*, as suas rotas são consideradas como impossíveis de cobrir naquela interface, por não haver nenhum *packet flow switch* naquele *datacenter* (esta informação encontra-se no campo **packet_brokers** do ficheiro de configuração) inviabilizando a utilização da interface para uma possível solução. Sempre que é adicionada uma rota válida é incrementada a frequência da rota numa *hash* que tem como propósito guardar as frequências. A rota é ainda adicionada a uma *hash* que guarda as rotas de cada tipo de serviço.

Cada elemento da lista de interfaces é uma instância de uma classe que representa uma interface, com os seguintes atributos:

- ***Hostname***;
- **Nome**;
- **Localização**;
- **Se está ou não coberta** (atributo utilizado pelo algoritmo do PlacementEngine);
- **Velocidade** (Gbps);
- **Rating**;
- **Rotas**;
- **Número de conexões**;

4.2 PlacementEngine

A função **save_graph_data** do DiscoveryEngine coloca todas as interfaces numa estrutura de dados em memória, partilhada entre os dois componentes, a **interface_db**.

O algoritmo descrito na Secção 3.2.2 foi implementado utilizando a interface_db como lista de interfaces válidas descobertas que podem ser consideradas para uma solução. As operações de cálculo dos *ratings* são feitas sobre as interfaces que compõem a interface_db.

Depois de identificar todas as soluções válidas que cumprem os requisitos mínimos de cobertura, é necessário apresentar os resultados do PlacementEngine ao utilizador do sistema. Os dados apresentados são:

1. Rotas que não podem ser cobertas porque apenas passam por datacenters em que não há *packet flow switches*.
2. No caso de não haver portas suficientes nos *packet flow switches* para implementar as soluções encontradas é indicado o *datacenter* com falta destes equipamentos.
3. Conjunto de interfaces que compõem a solução com melhor cobertura (última solução identificada), bem como o relatório e o mapa de rede com os *splitters* (a azul).
4. Conjunto de interfaces que compõem a solução com o melhor rácio cobertura-custo, o relatório e o mapa de rede com os *splitters*.

Os relatórios são compostos pelos elementos indicados na Secção 3.2.3 e foram criados com o Prawn [27], uma biblioteca Ruby de geração de PDFs. Os relatórios gerados apresentam uma estrutura igual à Figura 4.11. À semelhança do mapa de rede criado no final do DiscoveryEngine os mapas de rede criados por este componente são gerados com o Graphviz.

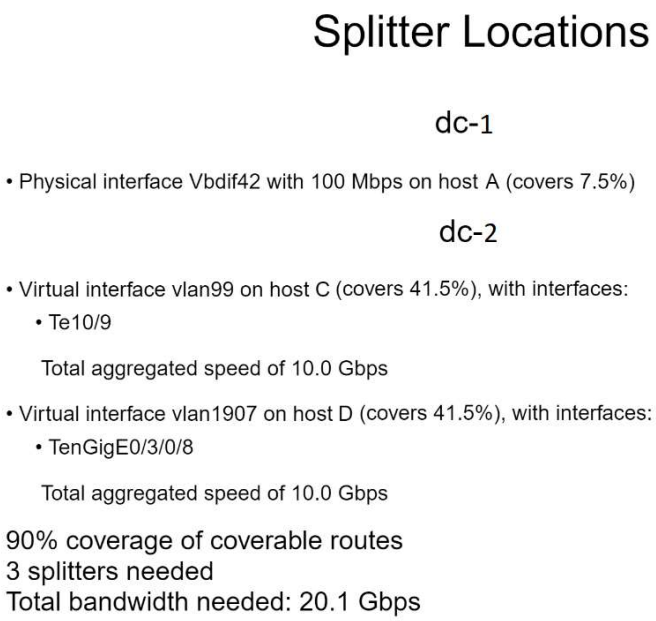


Figura 4.11: Exemplo do relatório gerado

4.3 Conclusão

Este capítulo descreveu a implementação do sistema que mapeia as rede de acesso a servidores internos e determina os pontos ótimos para a colocação de sensores que permitam a captura e respetiva inspeção do tráfego das redes mapeadas.

A implementação do sistema foi repartida em dois componentes fulcrais. O DiscoveryEngine, componente responsável pelo mapeamento das redes de acesso aos servidores e o PlacementEngine, o componente incumbido de determinar os pontos para a colocação de sensores na rede mapeada.

A implementação do processo de descoberta de caminhos de rede é explicada em detalhe para que seja possível entender como os elementos da rede são descobertos e como representam corretamente a rede física mapeada.

Já a implementação do algoritmo de determinação de pontos ótimos para a colocação de sensores aponta para o algoritmo detalhado na Secção 3.2.2, sendo assim uma implementação direta do algoritmo, não sendo preciso uma explicação detalhada da sua implementação.

O capítulo seguinte demonstra os resultados obtidos através de um conjunto de testes feitos ao sistema. Isto para que seja possível perceber se o sistema cumpre realmente o seu propósito, a determinação de pontos para a colocação de sensores em redes complexas.

Capítulo 5

Resultados e Avaliação

Os capítulos prévios descreveram em detalhe o objetivo, a arquitetura e respetiva implementação do NIDSPlacer. Sendo assim, é fundamental ter um capítulo que avalie e analise o sistema desenvolvido, de modo a ser possível perceber se os objetivos pretendidos foram alcançados, e que todos os requisitos da secção 3.1 foram satisfeitos ou pelo menos tidos em consideração.

Este sistema foi desenvolvido de forma a preencher um conjunto de imperfeições no processo de determinação dos pontos de inserção de sensores na rede interna da MEO, tornando o processo bastante ineficiente, resultando depois numa solução de monitorização, possivelmente, pouco eficaz. A solução atual para a determinação destes pontos é um procedimento manual, feito idealmente por um conjunto de pessoas, demorado e complexo. A metodologia aplicada pela MEO para a determinação dos pontos não é conhecida na altura da redação deste documento porque envolve um processo demorado e que se encontra atualmente em desenvolvimento. Como tal, não é possível comparar rigorosamente o sistema desenvolvido com todo o processo manual atualmente aplicado para determinar os pontos de inserção de sensores.

Este capítulo apresenta uma análise minuciosa aos resultados obtidos através de testes feitos ao sistema desenvolvido. Os testes realizados estão relacionados com desempenho, eficácia e eficiência.

Todos os testes foram realizados numa máquina com o sistema operativo Ubuntu 18.04 LTS, 16GB de RAM e um processador Intel i7-8550U com 4 *cores* (8 *threads*, 2 *threads* por core). As redes de utilizadores da MEO disponíveis para testes foram: a rede VPN acessos remotos, uma rede interna por cabo e uma rede Wi-Fi com acesso à Internet (sem acesso direto à rede interna). Os únicos equipamentos mapeados nestes testes foram aqueles pertencentes a redes de *datacenter*. Esta configuração do sistema tem como propósito demonstrar um caso real de utilização em que os sensores apenas podem ser colocados em *datacenters*.

Este capítulo divide-se em várias secções. A Secção 5.1 avalia o DiscoveryEngine do ponto de vista do desempenho esperado e da sua escalabilidade. A secção seguinte,

Secção 5.2, avalia o PlacementEngine através da análise assintótica do algoritmo desenvolvido, bem como através de testes que permitem uma avaliação empírica deste componente. A Secção 5.3 faz uma avaliação do sistema como um todo, dados diferentes critérios como a compatibilidade, usabilidade, funcionalidade e a comparação de desempenho da solução manual vs automatizada.

5.1 Avaliação do DiscoveryEngine

Esta secção expõe a bateria de testes feitos ao DiscoveryEngine, bem como a avaliação deste componente com base nos testes realizados.

O DiscoveryEngine é um componente com uma arquitetura *multi-thread*, como tal é importante perceber qual o impacto da escolha desta arquitetura face a uma *single-thread*. Foram feitos testes com três configurações diferentes: 1 *thread* (*single-thread*), 4 *threads* e 8 *threads* (configuração correspondente ao número de *threads* simultâneas suportadas pelo processador). Cada configuração do número de BuildThreads foi por sua vez testada com três configurações diferentes do sistema: 1 *datacenter* e 1 serviço (email) - 34 servidores, 1 *datacenter* e 3 serviços (email, DNS e AD) - 66 servidores, e 2 *datacenters* e 3 serviços (email, DNS e AD) - 138 servidores. Estes testes foram feitos com apenas uma rede, a de acessos remotos.

Os resultados obtidos com estes testes encontram-se apresentados na Tabela 5.1 e Figura 5.1.

Número de *threads*	**1 DC 1 Serviço**	**1 DC 3 Serviços**	**2 DC 3 Serviços**
1 *thread*	5.69 min	12.767 min	25.066 min
4 *threads*	1.81 min	4.655 min	7.329 min
8 *threads*	1.036 min	3.205 min	4.318 min

Tabela 5.1: Tempo, em segundos, demorado pelo DiscoveryEngine

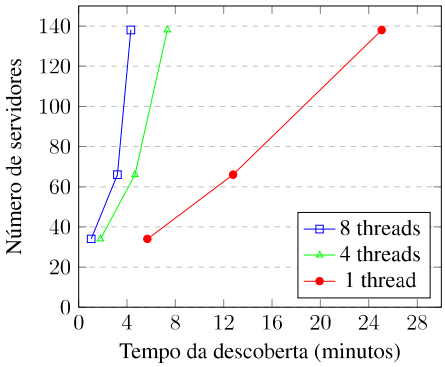


Figura 5.1: Gráfico com o tempo de descoberta para diferentes configurações

Através da análise dos valores da Tabela 5.1, que se encontram retratados na Figura

5.1, é possível perceber a ampla diferença de desempenho entre uma arquitetura *multi-thread* e uma *single-thread*. O tempo de execução pode ser até 6 vezes menor com 8 *threads*, e 4 vezes menor com 4 *threads*. Este ganho de desempenho é chamado de *speedup*. Já a diferença entre 4 e 8 *threads* é mais reduzida, podendo diminuir para metade o tempo de descoberta se forem utilizadas 8 *threads*, o que se justifica com a aproximação do número de *threads* concorrentes suportada pelo processador. As diferenças encontradas são facilmente explicáveis devido ao aproveitamento da arquitetura *multi-thread* que se encontra nos processadores mais modernos, sendo possível executar várias tarefas paralelamente, reduzindo assim o tempo de execução de qualquer programa, desde que os mecanismos de concorrência necessários sejam implementados corretamente.

Os outros testes feitos ao DiscoveryEngine têm como objetivo avaliar a escalabilidade do módulo. Para esse fim foi adicionada mais uma configuração aos testes, para além da configuração já existente com apenas uma rede, com três redes de utilizadores: VPN, Wi-Fi e a rede por cabo. Foram estendidos os testes feitos anteriormente com a adição de mais duas configurações de serviços: 3 *datacenters* e 3 serviços - 169 servidores e 3 *datacenters* e 6 serviços - 212 servidores. Perfazendo no total cinco casos para cada teste com os conjuntos de redes definidas. Para cada conjunto de redes os testes utilizaram uma configuração de 8 *threads*, de modo a tirar partido da arquitetura *multi-thread*.

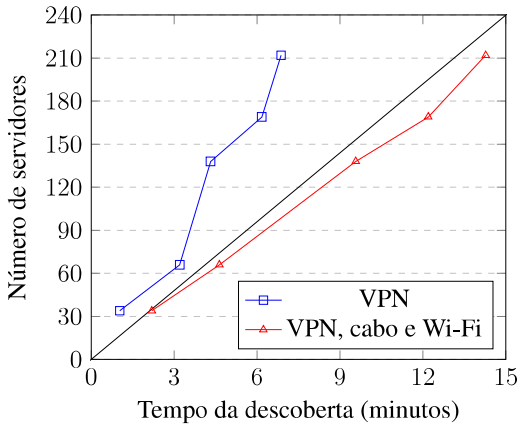


Figura 5.2: Tempo de descoberta para diferentes redes

Com os dados apresentados na Figura 5.2 é possível constatar a escalabilidade do sistema para diferentes cargas/*inputs*, desde uma rede e 34 servidores até três redes e 212 servidores. À medida que a carga do sistema vai aumentando o tempo de descoberta varia de forma aproximadamente linear face à carga do sistema, o que demonstra a escalabilidade do mesmo. Isto deve-se à natureza do algoritmo de descoberta que tira partido das *caches* implementadas que previnem a repetição de pedidos já efetuados. Outro facto que se relaciona diretamente com a escalabilidade do sistema é o facto dos acessos a servidores diferentes partilharem ligações L2 e/ou equipamentos de rede da mesma infra-

estrutura física. Desse modo o número de equipamentos de rede não tem grande variação, ao contrário do número de servidores, como se pode ver na Figura 5.3, podendo assim tirar partido das *caches* com informação dos equipamentos de rede para cada pedido de obtenção do caminho de rede para um determinado servidor.

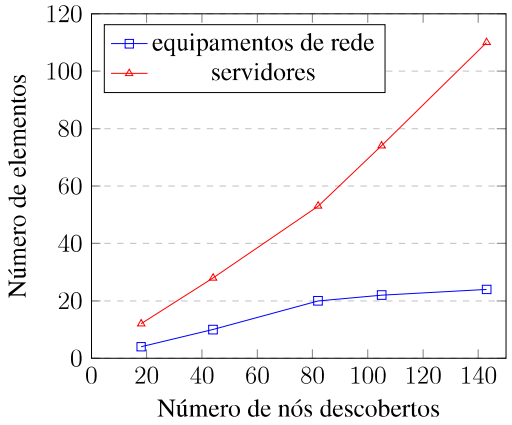


Figura 5.3: Crescimento do número de equipamentos de rede e servidores (VPN)

Servidores identificados	Nós	Equipamentos de rede	Servidores contactáveis
34	18	4	12
66	44	10	28
138	82	20	53
169	105	22	74
212	143	24	110

Tabela 5.2: Número de nós identificados - VPN

Servidores identificados	Nós	Equipamentos de rede	Servidores contactáveis
34	18	4	12
66	50	10	29
138	86	21	55
169	111	24	77
212	150	26	110

Tabela 5.3: Número de nós identificados - VPN, cabo e Wi-Fi

A variação do número de equipamentos de rede entre os resultados do teste com apenas uma rede de utilizadores, Tabela 5.2, e o teste com três redes, Tabela 5.3, é bastante reduzida o que indica a partilha da mesma infraestrutura física entre diferentes segmentos da rede. Neste caso o tráfego proveniente de diferentes redes de utilizadores partilha grande parte da mesma rede de *datacenter*. Estes locais são os que agregam mais tráfego, sendo portanto os pontos fulcrais da rede onde devem ser colocados sensores.

O número de nós identificados pode parecer baixo para uma rede de grande dimensão, mas este número é o resultado da filtragem de equipamentos de rede pelo DiscoveryEngine. Esta filtragem é feita para que apenas sejam incluídos os equipamentos de rede considerados pertinentes pelo utilizador do sistema, sendo neste caso os únicos equipamentos pertinentes os de *datacenter*.

Dada a complexidade da rede da MEO todos os testes feitos são representativos do comportamento que o NIDSPlacer teria numa outra rede de grande dimensão e complexidade.

5.2 Avaliação do PlacementEngine

Esta secção tem como finalidade a avaliação do PlacementEngine com base na análise assintótica ao algoritmo responsável por determinar os pontos para colocação de sensores e na avaliação empírica do desempenho deste componente. A análise teórica do algoritmo foi feita para ser possível perceber o comportamento do algoritmo para volumes de dados de grande dimensão que apenas podem ser observados em cenários utópicos, ao contrário dos volumes de dados observados nos testes feitos.

A análise assintótica de um algoritmo pode ser feita com o uso da notação big O para indicar o limite superior da complexidade temporal ou espacial, *i.e* como o tempo de execução do algoritmo ou a memória consumida crescem consoante o tamanho do *input*. A notação big Ω tem o mesmo propósito para indicar o limite inferior da complexidade. Para o caso normal ou esperado pode ser também utilizada a notação big O.

A complexidade de um algoritmo pode ser definida para três casos: pior, esperado e melhor. Estes casos variam consoante o *input* do algoritmo. O pior caso é a única solução válida encontrada ser composta por todas as interfaces válidas. O caso esperado é uma solução aproximada da ótima. Sendo a heurística criada baseada na *greedy* seria plausível assumir que o seu desempenho é equivalente à heurística *greedy* básica, *i.e* não será pior que um fator de $ln\ n$ em comparação com a solução ótima [24], mas como a heurística/*rating* criado tem componentes únicos não é possível fazer essa equivalência. Dessa forma não há uma estimativa teórica para a complexidade do caso esperado. Esses componentes únicos do *rating* aplicado fazem com que as soluções determinadas pelo algoritmo se encontrem mais próximas da solução ótima dadas as características particulares da colocação de sensores numa rede de computadores. Por último o melhor caso é encontrar uma só interface que cubra todas as rotas, garantindo assim uma cobertura máxima das rotas.

O n representa o número de interfaces válidas e o m o número de rotas cobertas por cada interface.

A análise da complexidade do algoritmo, apresentado no Anexo A.1, foi feita com base nas seguintes operações do algoritmo:

- O ciclo *while* da linha 6 é executado n vezes para o pior caso, porque a solução encontrada é composta por todas as interfaces. Para o melhor caso não há uma garantia do número de iterações devido à fórmula de cálculo dos *ratings*, mas pode-se assumir que é executado aproximadamente $\frac{n}{2}$ vezes (esta aproximação surgiu com base no número de iterações calculado nos testes, referidos mais adiante nesta secção). Já o melhor caso vai ter apenas uma iteração porque apenas vai cobrir uma interface.

- A função da linha 7, correspondente ao Anexo A.2, tem uma complexidade de O(nm) para todos os casos, por necessitar de percorrer todas as interfaces para calcular o *rating* de cada uma, com base nas rotas que cobrem.

- O ciclo *for* da linha 12 tem uma complexidade de O(m), e tem como propósito indicar que as rotas da interface da iteração atual foram cobertas pela solução atual do algoritmo.

- Todos os outros passos do algoritmo apresentam uma complexidade constante, como tal não entram para os cálculos da complexidade. O ciclo *for* da linha 22 está relacionado com a validação da cobertura de cada um dos serviços e não entra diretamente para o processo de determinação da solução de cobertura definida pelo algoritmo.

O algoritmo desenvolvido e apresentado nos anexos A.1 e A.2 tem a complexidade temporal especificada na Tabela 5.4.

Pior caso	**Caso esperado**	**Melhor caso**
O($n(nm+m)$)$\Leftrightarrow$ O(n^2m+nm)	O($\frac{n}{2}(nm+m)$)$\Leftrightarrow$ O($\frac{n^2m+nm}{2}$)	$\Omega(nm+m)$

Tabela 5.4: Complexidade temporal do algoritmo descrito em A.1

Todas as complexidades temporais da Tabela 5.4 foram calculadas com base na complexidade temporal de cada passo do algoritmo, sendo alguns passos descartados devido à falta de peso para o cálculo da complexidade.

Como se pode verificar a complexidade temporal do algoritmo está dependente do cálculo de *ratings* e do número de iterações feitas para achar a solução que garante a cobertura máxima. A complexidade temporal do algoritmo pertence à classe das funções quadráticas, podendo a complexidade ser descrita por O(n^2m), descartando os termos de ordem menor e a constante $\frac{1}{2}$ do caso esperado. A avaliação empírica pode indicar, com maior assertividade, a classe de funções que melhor descreve o algoritmo criado para os casos esperados.

Os gráficos da Figura 5.4 representam o tempo de execução do PlacementEngine e as iterações do algoritmo para o mesmo volume de interfaces. Apesar das redes utilizadas para testes serem compostas no máximo por 150 nós, as interfaces válidas não passaram das 27. Dessa forma os resultados dos testes podem não transparecer a realidade da rede, já que há bastantes interfaces que não foram identificadas por falta de informação ou por pertencerem a interligações com redes não mapeadas. Devido à dependência em relação às plataformas e rede interna da MEO, não é possível ter resultados de uma rede com outra composição e que pudesse apresentar resultados mais completos. Não é possível validar se todas as redes corporativas, ou com estrutura semelhante, iriam ou não ter apenas algumas dezenas ou centenas de interfaces válidas para serem analisadas pelo algoritmo do PlacementEngine. Se todas as redes, dadas as configurações de teste (apenas a rede de *datacenter* a ser mapeada), forem compostas por dezenas ou centenas de interfaces válidas o desempenho do algoritmo é suficientemente bom.

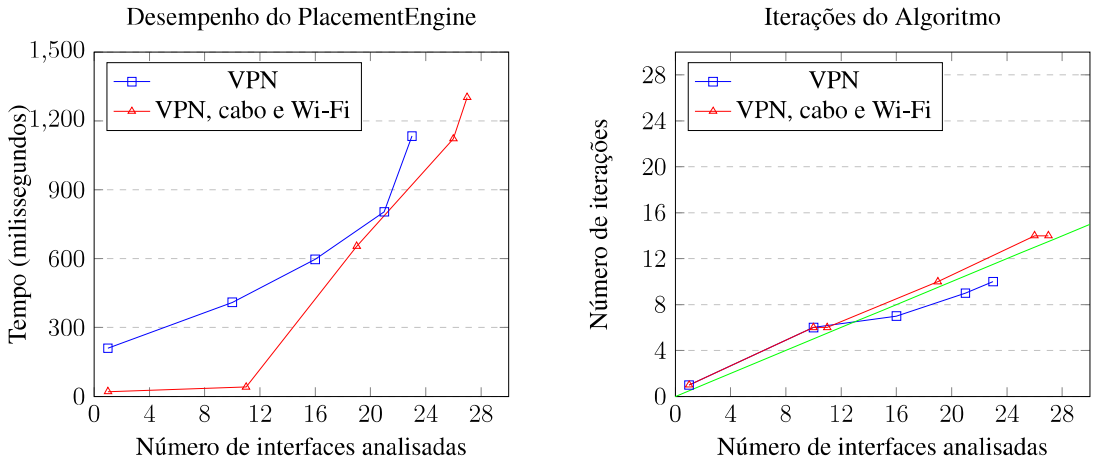


Figura 5.4: Gráficos do PlacementEngine

O gráfico da Figura 5.4, referente ao tempo de execução do PlacementEngine, permite tirar ilações em relação à complexidade do algoritmo desenvolvido. A curvatura de ambas as funções do gráfico é similar a uma função quadrática, neste caso n^2, o que valida a análise assintótica feita.

O número de iterações parece seguir uma distribuição linear com declive 1/2, a verde na figura, indicando assim que o número de iterações é aproximadamente metade do número de interfaces, daí o $\frac{n}{2}$ referido na complexidade do caso esperado do algoritmo. Para um maior número de interfaces não é possível saber se este facto se mantém, ou se a distribuição será mais próxima de uma logarítmica, devido à agregação de tráfego em pontos ainda mais centrais resultando assim num menor número de iterações e de interfaces que compõem a solução.

Apesar de ter uma complexidade temporal quadrática o algoritmo desenvolvido continua a ser uma solução melhor, com melhor desempenho, do que um algoritmo de procura

exaustiva que apresenta uma complexidade temporal de O($2^n m$). Isto porque uma solução de procurar exaustiva necessita de gerar as combinações do conjunto de todas as interfaces válidas para depois saber quais soluções garantem os requisitos de cobertura mínima estipulados no ficheiro de configuração.

Apesar do algoritmo ser de natureza quadrática, o sistema como um todo é executado em alguns minutos ou poucas horas, garantindo assim a compatibilidade do sistema com a janela de tempo, na ordem das semanas, indicada nos requisitos.

A forma como o algoritmo foi construído permite uma maior aproximação à solução ótima face ao algoritmo *greedy* simples, que apenas escolheria as interfaces que cobririam mais rotas. A aproximação á solução ótima deve-se ao cálculo do *rating* especificamente criado para o problema, *i.e* um *rating* composto por elementos que caracterizam as interfaces de uma rede física e a sua cobertura da rede.

5.3 Avaliação do Sistema

Esta secção tem o objetivo de avaliar o sistema como um todo, validando todos os requisitos necessários para satisfazer os critérios essenciais de uma solução adequada para o problema de colocação de sensores garantindo uma cobertura ótima da rede.

5.3.1 Compatibilidade

O requisito de compatibilidade não é totalmente possível de satisfazer visto que para diferentes organizações a forma como os metadados relativos a servidores e equipamentos de rede são armazenados é diferente. O sistema foi desenvolvido a pensar na rede interna da MEO e nos meios disponíveis para recolher informação sobre a rede interna. Isto não quer dizer que algumas alterações ao sistema não possam permitir a sua utilização noutra organização. Se por exemplo forem feitos *plugins* com a mesmas capacidades do NNMPlugin e ElasticPlugin o sistema irá funcionar exatamente da mesma forma noutra organização/rede.

5.3.2 Usabilidade

O sistema cumpre o requisito de usabilidade por ser simples e prático de usar, não necessitando de uma interface gráfica para ser utilizado devido à sua simplicidade de configuração e uso. O sistema pode ser simplesmente configurado e executado logo de seguida. O conjunto de configurações necessárias para colocar o sistema em funcionamento é claro, de fácil perceção e não muito extenso. O ficheiro de configuração é composto por sete campos com propósitos diferentes. Cada campo do ficheiro de configuração é por sua vez composto por mais alguns campos. Todos os campos são auto-explicativos, não

sendo precisa uma documentação muito detalhada. A grande maioria dos campos é apenas alterada uma vez e servem para vários tipos de casos de uso em que o sistema pode ser utilizado. A parte do ficheiro de configuração que vai variar mais é o conjunto de serviços que devem ser analisados pelo sistema.

Todos os elementos que compõem o *output* do sistema são percetíveis para qualquer utilizador. Tanto os relatórios gerados, como os mapas, são de fácil interpretação mesmo para utilizadores que não tenham um entendimento profundo dos componentes da rede e do seu funcionamento.

5.3.3 Funcionalidade

Do ponto de vista funcional o *output* esperado do sistema seria composto pelo número de sensores necessários, a sua localização física e a largura de banda necessária para processar o tráfego a ser capturado pelos sensores, para que depois se possa implementar corretamente o conjunto de NIDSs necessários para gerar os eventos de segurança.

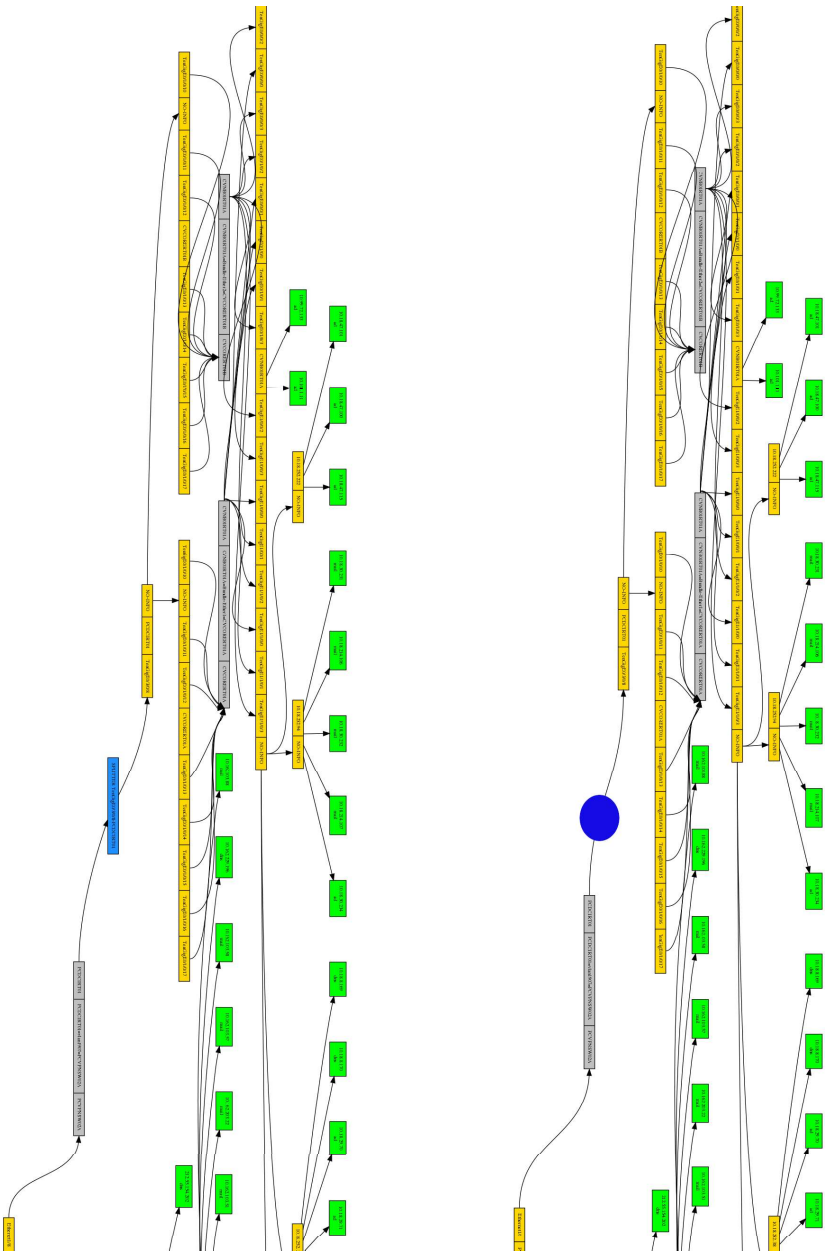
O *output* do sistema é composto por todos os elementos essenciais para a implementação da solução determinada pelo sistema. Os mapas da rede indicam o local físico onde devem colocados cada um dos sensores. Os relatórios, um para a solução que garante a cobertura máxima e outro para a solução com melhor custo-cobertura, completam a informação de cada mapa com a percentagem de rotas cobertas por cada interface, bem como a velocidade de cada. No final de cada relatório é especificado o número de sensores e a largura de banda, em Gbps (Gibabits por segundo), necessária para capturar todo o tráfego do conjunto de ligações associadas à colocação dos sensores.

5.3.4 Solução automatizada vs Solução manual

Uma forma de fazer uma comparação entre as duas soluções, manual e automatizada, é fazer o exercício de colocação manual de sensores e comparar o resultado com aquele gerado pelo sistema, baseado no mesmo mapa de rede.

A Figura 5.5 mostra a colocação dos sensores feita pelo sistema vs a colocação de sensores dado o critério de apenas serem cobertas as interfaces que cobrem mais rotas/-caminhos originários na parte inferior do mapa (*i.e* as outras redes não mapeadas ou nos equipamentos virtuais da VPN) com destino aos servidores identificados (parte superior do mapa), de modo a cortar ao mínimo o número de sensores garantindo uma cobertura total. Este mapa de rede tem apenas interfaces com velocidade inferior a 10 Gbps e os sensores considerados suportam velocidades até 40 Gbps. As interfaces escolhidas na solução manual representam a solução ótima porque as outras interfaces, que cobrem mais rotas, não podem ser cobertas fisicamente, ou por falta de informação, ou por pertencerem a redes não mapeadas. Ambos os mapas correspondem ao terceiro caso da tabela 5.2, *i.e* 2 *datacenters* e 3 serviços.

Como se pode comprovar pela observação da Figura 5.5, neste caso particular ambas as soluções levam a conjuntos iguais. Para redes de maior dimensão as soluções encontradas podem-se diferenciar já que o algoritmo desenvolvido não garante uma solução ótima mas sim uma aproximação. Nesse caso torna-se bastante complicado escalar o processo manual devido à complexidade da rede. Uma grande vantagem do uso do sistema desenvolvido face a um processo manual, é a filtragem das interfaces que não podem ser consideradas como elementos de uma solução (interfaces com velocidade superior à dos sensores ou às portas dos *packet flow switches*, interfaces de interligação com redes não mapeadas e interfaces sem mapeamento possível). Esse trabalho é feito pelo DiscoveryEngine e diminui bastante o volume de dados a ser processado pelo PlacementEngine, reduzindo assim o tempo de execução do sistema como um todo.



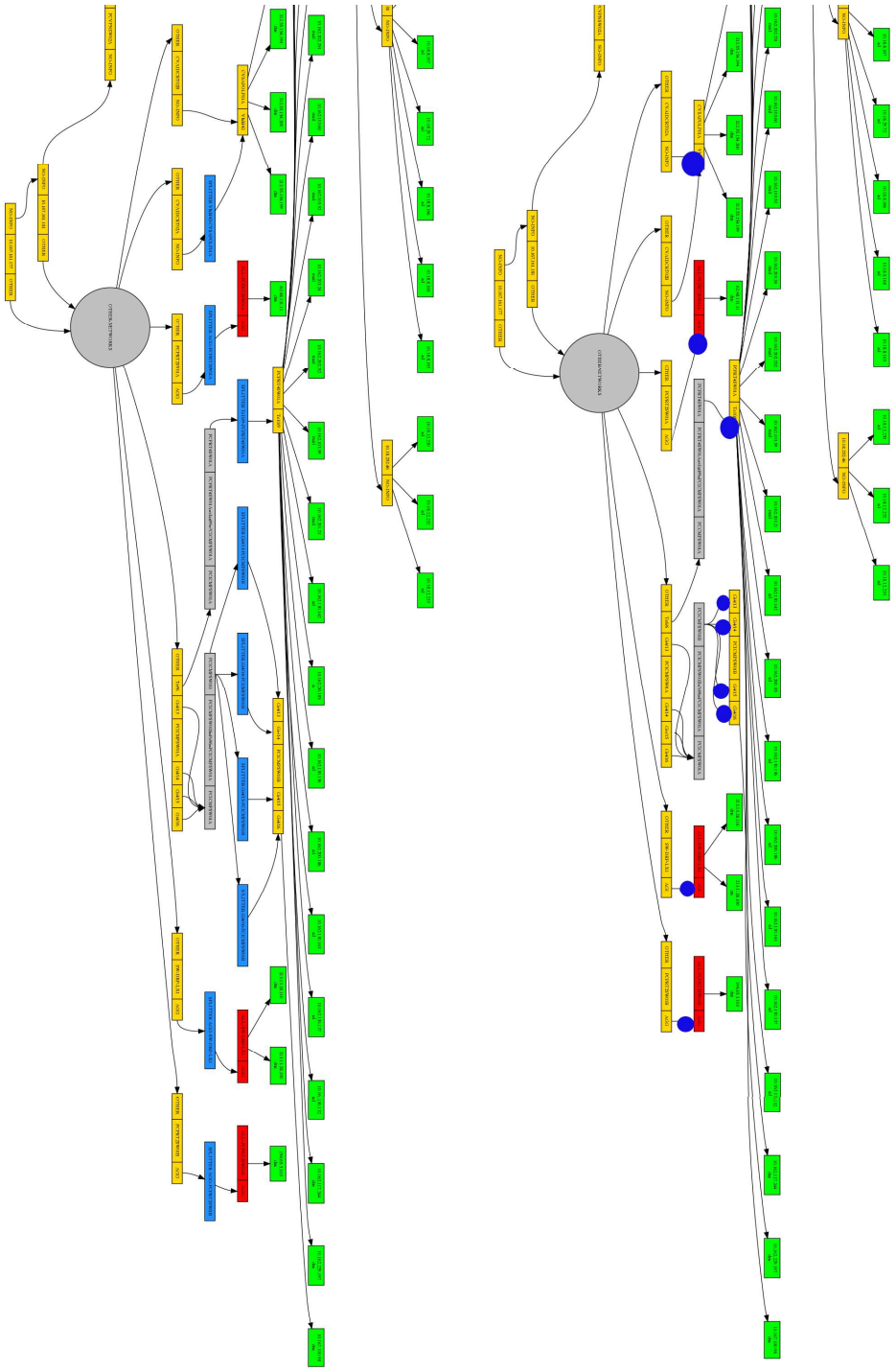


Figura 5.5: Solução automatizada vs manual

Apesar de não ser possível comparar, com dados concretos, o desempenho do sistema vs desempenho da solução manual, é possível fazer uma estimativa do tempo que uma solução manual demoraria. Uma solução manual para o problema da colocação de sensores numa rede complexa é uma tarefa para um grupo de pessoas e não é algo que seja feito em poucas horas, podendo-se estender por várias semanas. Para ter um melhor desempenho em relação à duração do processo de determinação dos pontos de colocação dos

sensores, o sistema automático teria apenas de ter tempos de execução inferiores a uma semana. Os resultados dos testes feitos nas secções 5.1 e 5.2 indicam que o sistema, configurado de forma a tirar partido das arquitetura *multi-thread* do processador, tem tempos de execução de 15 minutos e alguns segundos para 212 servidores e 3 redes.

Estima-se que estes tempos possam chegar a 1 hora para 1000 servidores e 3 redes (212 servidores descobertos em 14 minutos, dado o crescimento linear do tempo de descoberta com o número de servidores $\Longrightarrow 1000 \times 14/212 \approx 66\ min$). Para que o sistema demorasse uma semana a executar eram necessários 152640 servidores e 3 redes ($10080\ min \times 212/14 = 152640\ servidores$), o que parece ser um número um pouco irrealista. A diferença entre o desempenho da aproximação manual e aquela criada pelo sistema é drástica. Para além disso, os resultados de uma solução manual estão sempre dependentes do fator humano, que poderá ou não resultar em erros no resultado final, devido ao desconhecimento do processo feito pelo um humano e à incerteza em relação à qualidade do processo. Para o resultado de uma solução manual ser equiparável ao da solução automática é necessário um conjunto de pessoas com competências avançadas sobre redes, mais especificamente sobre a estrutura da rede interna e dos padrões de tráfego dessa rede, essa análise é complexa e torna o processo mais demorado.

Qualquer processo de determinação de pontos para a monitorização da rede é apenas feito anualmente, trimestralmente ou bimestralmente devido ao custo relacionado com a colocação de sensores na rede. Desse modo, o sistema desenvolvido poderia ter tempos de execução bem superiores aqueles observados nos testes e ainda assim estaria dentro do intervalo de tempos de execução aceitáveis para a periodicidade do processo.

A juntar aos tempos de execução do sistema bastante inferiores à solução manual, pode-se ainda juntar a melhoria de desempenho do ponto de vista qualitativo do processo. Apesar de não haver dados concretos que permitiam uma comparação entre as duas soluções, a aproximação encontrada pelo sistema automático vai beneficiar sempre dos dados recolhidos na fase de descoberta. Estes dados são recolhidos de forma automatizada e são retirados de fontes de dados verossímeis, dessa forma o algoritmo que determina os pontos de colocação dos sensores vai ter uma tarefa bastante facilitada para encontrar uma aproximação que cubra todo o tráfego. Um pormenor que facilita bastante a análise da rede é o facto das redes em que não devem ser colocados sensores não aparecerem no mapa de rede, assim não há elementos desnecessários que só dificultariam o processo de análise da rede feito pelo PlacementEngine.

A análise feita pelo PlacementEngine tira partido dos vários benefícios do DiscoveryEngine, assim o resultado final da solução automática é produzido com um esforço muito inferior a o de uma análise manual da rede. A qualidade do resultado da solução automática é equivalente à de uma solução manual que precisa de um grande esforço para ser realizada.

5.4 Conclusão

Apesar do NIDSPlacer não ter uma base de comparação empírica em relação a um processo manual que cumpre o mesmo propósito, este capítulo conseguiu mostrar melhorias a nível dos tempos de execução e esforço dispensado conseguidas através da automatização do processo.

Ambos os componentes principais do sistema desenvolvido foram examinados neste capítulo.

O DiscoveryEngine mostrou tirar partido da sua arquitetura *multi-thread* da melhor forma, reduzindo os tempos de execução com a utilização de várias *threads* ao invés de uma só *thread*. O DiscoveryEngine mostrou também ser escalável para diferentes cargas, centenas de servidores e uma só rede ou centenas de servidores e três redes diferentes. A escalabilidade deste componente deve-se à natureza do seu algoritmo de descoberta que tira partido das *caches* implementadas.

O PlacementEngine mostrou suportar o conjunto de testes feitos ao componente. As soluções determinadas pelo componente conseguem resolver o problema de colocação de sensores aproximando a solução encontrada da solução ótima que apenas é possível de determinar com uma procura exaustiva. Os tempos de execução do componente são na ordem dos milissegundos para as dezenas de interfaces analisadas nos testes feitos. Dessa forma é possível concluir que o objetivo de ter uma solução automatizada que tivesse tempos de execução inferiores a uma semana foi cumprido.

Capítulo 6

Conclusão

Neste trabalho, apresentámos o NIDSPlacer, uma solução automatizada que resolve o problema da determinação de pontos para a inserção de sensores numa rede. A solução usada anteriormente à criação do sistema proposto, era um processo manual, demoroso e complexo, que necessitava de recursos idóneos, muitas vezes difíceis de arranjar. Como se pode ver no Capítulo 5 o sistema proposto resolveu todas as limitações do processo manual. O processo automatizado desenvolvido é drasticamente mais rápido que o processo manual. A qualidade dos resultados do processo automatizado mostra que ele garante uma solução de monitorização totalmente eficaz e eficiente. Apesar de não haver termo de comparação entre os dois tipos de processo, manual e automatizado, a avaliação feita permite concluir que o desempenho do sistema criado garante o requisito do tempo de execução inferior a uma semana. A automatização do processo permite que os recursos que eram despendidos no processo manual possam ser utilizados noutras tarefas que precisem efetivamente de recursos humanos com conhecimento especializado da rede.

O sistema desenvolvido tem duas limitações. As soluções de cobertura da rede determinadas não são ótimas, mas sim aproximadamente ótimas, isto porque o algoritmo que determina os pontos de inserção dos sensores necessita de ser escalável, e o único algoritmo que garante a solução ótima não é escalável, porque necessita de fazer uma procura exaustiva para achar tal solução. A outra limitação tem haver com os equipamentos de rede descobertos na fase de descoberta. O NIDSPlacer é apenas capaz de identificar os equipamentos diretamente conectados aqueles que aparecem no *traceroute*, dessa forma podem existir lacunas nos mapas de rede, estas resultam na falta de equipamentos nos mapas criados. Esta limitação podia ser corrigida com a criação de um processo complexo, que permitiria saber por onde circulava o tráfego de/para pelos equipamentos de L2, que não aparecem no *traceroute*, enriquecendo assim o processo de descoberta já implementado, de modo a completar o mapa de rede com toda a informação possível.

Os resultados apresentados pelo sistema mostram que pode e deve ser considerado como solução viável para o problema da colocação de sensores na MEO, e em outras organizações, com algumas modificações aos *plugins* criados. Tanto o seu desempenho

como a qualidade do resultado final apresentado são bastante melhores que qualquer processo manual equivalente. Os resultados são promissores, mas no futuro, deve ainda ser feita uma avaliação que compare o resultado e desempenho da solução manual em comparação com o sistema automatizado. Para além desta comparação devem ainda ser feitos testes noutras redes, de outras organizações, de modo a comprovar a aplicabilidade do sistema desenvolvido num contexto diferente, fazendo as alterações necessárias ao sistema.

Apêndice A

Algoritmos

A.1 Procura de soluções ótimas

Algoritmo descrito na próxima página

```
1  Function run_optimal_interface_search(interfaces, sensor_speed,
     route_frequencies, min_coverage, min_service_coverages, routes,
     service_routes, services, packet_switch_port_speed):
2      sols, covered_interfaces, covered_routes, covered_service_routes,
3          coverages = {}
4      num_sensors, current_coverage = 0
5      met_minimum_requirements = false
6      while current_coverage < 1 do
7          interface = run_rating_calculation(interfaces, route_frequencies,
             covered_routes, covered_service_routes, sensor_speed,
             packet_switch_port_speed)
8          interface.set_covered
9          interface.rating = 0
10         num_sensors += interface.connections
11         covered_interfaces.add(interface)
12         foreach route in interface.routes do
13             covered_routes.add(route)
14             destination = get_destination(route)
15             service = get_destination_service(destination)
16             covered_service_routes.get(service).add(route)
17         end
18         interfaces.remove_interface(interface)
19         current_coverage = covered_routes.size/routes.size
20         if current_coverage ≥ min_coverage and !met_minimum_requirements then
21             done = true
22             foreach service in services do
23                 service_coverage = covered_service_routes.get(service).size /
                     service_routes.get(service).size
24                 if service_coverage < min_service_coverages.get(service) then
25                     done = false
26                     break
27                 else
28                     coverages.add(service_coverage)
29                 end
30             end
31             if done then
32                 met_minimum_requirements = true
33             end
34         end
35         if met_minimum_requirements then
36             coverages.add(current_coverage)
37             sols.add(covered_interfaces, coverages, num_sensors)
38             coverages = {}
39         end
40     end
41 return sols
42
```

A.2 Cálculo dos *ratings* de cada interface

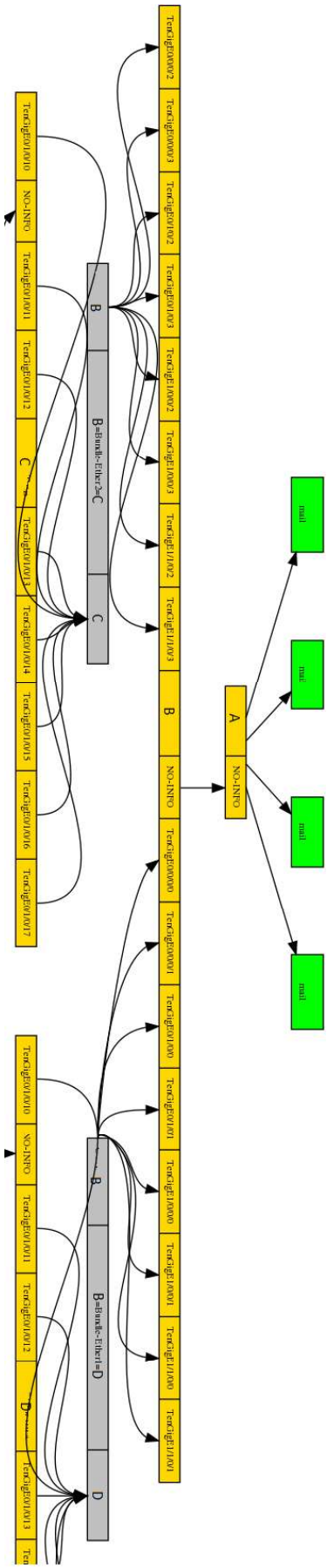
```
1  Function run_rating_calculation(interfaces, route_frequencies,
     covered_routes, covered_service_routes, packet_switch_port_speed,
     sensor_speed):
2      result = null
3      max_rating = 0
4      foreach interface in interfaces do
5          if interface.speed > sensor_speed or
6              interface.speed > packet_switch_port_speed then
7              next
8          end
9          count_uncovered, element_frequency = 0
10         foreach route in interface.routes do
11             if covered_routes.get(route) == null then
12                 count_uncovered += 1
13                 destination =get_destination(route)
14                 service =get_destination_service(destination)
15                 element_frequency += 1 / route_frequencies.get(route)×
16                     covered_service_routes.get(service)
17             end
18         end
19         interface.rating = count_uncovered × element_frequency×
20             (interface.connections / interface.speed)
21         if interface.rating > max_rating then
22             max_rating = interface.rating
23             result = interface
24         end
25     end
26 return result
```

Apêndice B

Outputs

B.1 Mapa de rede (exemplo)

Imagem na próxima página


A
NO-INFO
mail
mail
mail
mail
B
NO-INFO
TenGigE0/0/0/0
TenGigE0/0/0/1
TenGigE0/1/0/0
TenGigE0/1/0/1
TenGigE1/0/0/0
TenGigE1/0/0/1
TenGigE1/1/0/0
TenGigE1/1/0/1
B
B=Bundle-Ether1=D
D
TenGigE0/1/0/10
NO-INFO
TenGigE0/1/0/11
TenGigE0/1/0/12
TenGigE0/1/0/13
B
B=Bundle-Ether2=C
C
TenGigE0/1/0/0
NO-INFO
TenGigE0/1/0/11
TenGigE0/1/0/12
TenGigE0/1/0/13
TenGigE0/1/0/14
TenGigE0/1/0/15
TenGigE0/1/0/16
TenGigE0/1/0/17
TenGigE0/0/0/2
TenGigE0/0/0/3
TenGigE0/1/0/2
TenGigE0/1/0/3
TenGigE1/0/0/2
TenGigE1/0/0/3
TenGigE1/1/0/2
TenGigE1/1/0/3

Abreviaturas

AD Active Directory.

ARP Address Resolution Protocol.

BD Base de Dados.

BGP Border Gateway Protocol.

CDP Cisco Discovery Protocol.

DNS Domain Name System.

EIGRP Enhanced Interior Gateway Routing Protocol.

EIT Equipamentos para Inspeção de Tráfego.

ICMP Internet Control Message Protocol.

IGRP Interior Gateway Routing Protocol.

IOC Indicator of Compromise.

IP Internet Protocol.

IPSec Internet Protocol Security.

L2 *Layer* 2 - camada de ligação de dados no modelo OSI [11].

L3 *Layer* 3 - camada de rede no modelo OSI [11].

LAN Local Area Network.

LLDP Link Layer Discovery Protocol.

MAC Media Access Control.

MIB Management Information Base.

NIDS Network Intrusion Detection System.

NNM Network Node Manager.

OID Object Identifier.

OSPF Open Shortest Path First.

RIP Routing Information Protocol.

SIEM Security Information and Event Management.

SNMP Simple Network Management Protocol.

SOAP Simple Object Access Protocol.

TTL Time to Live.

VLAN Virtual Local Area Network.

VPN Virtual Private Network.

WSDL Web Service Definition Language.

Bibliografia

[1] Savon version 2.0. http://savonrb.com/version2/, 2012. [Online].

[2] Cisco ASA Series General Operations CLI Configuration Guide - Chapter: SNMP. https://www.cisco.com/c/en/us/td/docs/security/asa/asa95/configuration/general/asa-95-general-config/monitor-snmp.html, 2018. [Online].

[3] Cve-2017-0144. https://nvd.nist.gov/vuln/detail/CVE-2017-0144, 2018. [Online].

[4] Graphviz - Graph Visualization Software. https://www.graphviz.org/, 2018. [Online].

[5] Reverse Path Filtering. http://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.kernel.rpf.html, 2018. [Online].

[6] traceroute(8) - Linux man page. https://linux.die.net/man/8/traceroute, 2019. [Online].

[7] Longe Olumide Babatope, Lawal Babatunde, and Ibitola Ayobami. Strategic sensor placement for intrusion detection in network-based ids. *International Journal of Intelligent Systems and Applications*, 6(2):61, 2014.

[8] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[9] Stephen P Borgatti. Centrality and network flow. *Social networks*, 27(1):55–71, 2005.

[10] James Britt and Neurogami. Queue. https://ruby-doc.org/core-2.5.0/Queue.html, N.D. [Online].

[11] John D Day and Hubert Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[12] Elastic. Elasticsearch. https://github.com/elastic/elasticsearch-ruby, 2019. [Online].

[13] Hewlett Packard Enterprise. *Developer's Toolkit Guide - Software Version 10.30*. HPE.

[14] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, pages 251–262. ACM, 1999.

[15] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[16] National Initiative for Cibersecurity Careers and Studies (NICCS). A Glossary of Common Cybersecurity Terminology. `https://niccs.us-cert.gov/about-niccs/glossary`, 2018. [Online].

[17] Apache Software Foundation. Apache lucene - query parser syntax. `https://lucene.apache.org/core/2_9_4/queryparsersyntax.html`, 2006. [Online].

[18] Felipe Grando, Diego Noble, and Luis C Lamb. An analysis of centrality measures for complex and social networks. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[19] Da-Yu Kao and Shou-Ching Hsiao. The dynamic analysis of wannacry ransomware. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 159–166. IEEE, 2018.

[20] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[21] James F Kurose. The link layer: Links, access networks, and lans. In *Computer networking: A top-down approach featuring the internet, 6th Edition*, chapter 5, pages 433–500. Pearson Education, 2012.

[22] James F Kurose. The network layer. In *Computer networking: A top-down approach featuring the internet, 6th Edition*, chapter 4, pages 305–412. Pearson Education, 2012.

[23] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[24] Dave Mount. Cmsc 451: Lecture 9 greedy approximation: Set cover. http://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect09-set-cover.pdf, 2017.

[25] Steven Noel and Sushil Jajodia. Attack graphs for sensor placement, alert prioritization, and attack response. In *Cyberspace Research Workshop*, pages 1–8, 2007.

[26] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[27] Prawn. Prawn: Fast, Nimble PDF Generation For Ruby. `https://github.com/prawnpdf/prawn`, 2019. [Online].

[28] Vijay V Vazirani. *Approximation algorithms*, pages 15–26. Springer Science & Business Media, 2013.